An artificial neural network based incremental placer for wire length reduction

by

Gagandeep

Submitted in partial fulfilment of the requirements

for the degree of Master of Technology in Electronics and Communication Engineering

Under the supervision of

Dr. Sneh Saurabh

Certificate

This is to certify that the thesis titled **"An artificial neural network based incremental placer for wire length reduction"** submitted by **Gagandeep** to the Indraprastha Institute of Information Technology Delhi, for the award of the Master of Technology, is an original research work carried out by him under my supervision. In my opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

August, 2022

Supervisor Name: Dr. Sneh Saurabh
Department of Electronics and Communication Engineering
Indraprastha Institute of Information Technology
New Delhi, 110020

Abstract

The total wire length of a design affects various QoR measures such as timing, power, and routability. Hence, it is a critical parameter to optimize in physical design. Post global routing, the total wire length can be further minimized by incremental placement, i.e., moving the cell instances by a small amount to reduce the total wire length. Traditionally, incremental placement is done by deterministically finding suitable locations for a cell in its vicinity. Subsequently, the wire length for these locations is estimated by performing global routing, and the move is accepted if it reduces the wirelength. However, most such moves are rejected due to rout ability constraints, and the time spent in finding suitable locations is wasted. As an alternative, we can randomly try locations in the vicinity of a cell and accept the movement if it reduces the wire length. This method will allow us to perform many such evaluations quickly. However, this method will have a low acceptance rate.

In this work, we propose a fast Artificial Neural Network (ANN) based incremental placement technique. We use an ANN to predict a good location for a cell using various input features. An ANN can predict a movement very quickly, and we can try many such movements in given time duration. Further, the acceptance rate for an ANN-based incremental placer can be enhanced by employing a high-quality training set. In this work, we generate training data set synthetically by choosing the right balance of randomness and constraint. We obtain training data by first generating placement and congestion scenarios for a small grid using a constrained-random strategy. Then, we determine the optimal solution for a cell in the generated problem. We refer to this approach of training data generation as solution-directed. Further, we augment the training data set by finding optimal problem-solution pairs by starting with a solution and then computing a problem for which the given solution is guaranteed to be optimal. We refer to this approach of training data generation as problem-directed. We demonstrate that we can train

an ANN efficiently by utilizing a mix of problem-directed and solution-directed training sets. We observe that an incremental placer utilizing ANN trained with mixed training data set produces around 15% more acceptance rate than using either problem-directed or solution-directed training data set. Further, we benchmark the proposed incremental placer against a random movement-based incremental placer on ICCAD2020 benchmark designs [1]. For a given time limit of three hours, the number of trials that the ANN-based incremental placer performs is of the same order as the random placement-based incremental placer. Further, the proposed incremental placer has an acceptance rate of approximately 5X of the random placement-based incremental placer. The wire length reduced per accepted movement for the ANN-based placer is approximately 20% higher than the random placement-based algorithm. Thus, the number of feasible solutions and their quality is significantly better than the random incremental placer. Overall, the runtime of the proposed placer is similar to the random incremental placer but produces a wire length reduction of 3x more than the random incremental placer. Hence, the results demonstrate that the proposed incremental placer can provide a good tradeoff between accuracy and runtime over the traditional incremental placer.

Acknowledgements

The work for this thesis was carried out at IIIT-Delhi, India, during the year 2021-2022. First and foremost, I would like to express my sincere gratitude to my research advisor Dr. Sneh Saurabh for regular guidance and encouragement throughout the journey of this work. Without his support, this work would never have been a successful one. My sincere thanks also goes to all the supporting staff for their prompt help and support whenever needed.

This work is dedicated to almighty and to the loving memory of my grand father. I am grateful to my grand father who always motivated and inspired me for working honestly in every field of life.

I am also thankful to my project partner Pranav Jain who was available for support whenever needed without which this work would not have been possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

To arrive at the final layout of the design representing the digital circuit required for the desired functionality, we perform the physical design step of the VLSI Design flow. The gate level netlist defining the connections of the logic gates and the sequential elements of the circuits is implemented in form of transistors and metal wires in the layout of the design which is used by the foundry for fabricating the design. For placing these logic gates on the design, we perform the placement step in two parts, global and detailed placement. After the global placement, we have a fair idea of the overall wire length of the design.

At this stage, we can move the cells or change the placement of the logic gates by small incremental changes on the layout to optimize the overall design wire length. Wire length is an important metric to optimize as it affects various QORs such as power, area, and timing in the final design layout. This step is called Incremental Placement. We perform only small changes in the incremental placement step so as to avoid changing the design placement solution completely. We work only on the small part of the design while performing the incremental placement because of this reason.

1.1 Incremental Placement in IC Design

There are various factors that affect the wire length of the design. Of these factors, congestion, placement, and connectivity are the major contributors. If these factors are effectively managed then the obtained physical design scenario can be called optimized in terms of wire length.

Generally, congestion in the grid represents the amount of space available for routing wires and placing the standard cells in certain areas of the design. Congestion results in detours and affects the wire length of the design in an adverse manner. When the placement of the design is changed, then nets connected to the cells are affected and the net length changes. Connectivity is useful information in the incremental placement since it indicates which nets would be affected if a cell is moved from one position to another in the design.

1.2 Motivation

Many conventional algorithms such as force-directed placement [2], simulated annealing [3] and analytical model-based placement [4] are capable of providing better placement of cells for optimized wire length. Also, these algorithms deterministically apply some heuristics to arrive at the better placement of the design while performing the incremental placement [5–12]. But, these algorithms involve complex mathematical computations which take a large run time. A run-time efficient algorithm for incremental placement could be an incremental placer that moves the cells to randomly selected positions and accepts the movements as a solution if the design wire length reduction results from the random movement. This is inherently fast as it does not involve any complex mathematical computations and heuristics. But, this is also a non-directed/convergent approach to arriving at a solution. This would not produce high-quality movements of the cells in design and also most of the movements would not result into lower wire length and hence would be discarded. This means this algorithm would have a low acceptance rate.

An algorithm involving ANN to produce the movement position for the cells

would have the per trial run time of the same order as that of the random position generation-based incremental placer. This is because the ANN involves very simple MAC operations for inference once trained using the relevant data. This solution produced by the machine learning model would be a directed approach to arriving at the solution if the data used for training the ANN model for solving the incremental placement problem is optimal and is able to model real case scenarios. Therefore the acceptance rate of this algorithm is expected to be very high as compared to the random position generator-based algorithm. This algorithm, therefore, is able to provide more total improvement as compared to the conventional algorithm of incremental placement. While performing incremental placement (incremental placement) in our work, we move one cell at a time and keep trying movement for different cells to minimize the overall design wire length. The run-time of the incremental placement algorithm can be expressed as: -

$$Runtime = N_{trials} \times T_{trial} \tag{1.1}$$

where, N_{trials} is the number of trials made for moving the cells in a design, and T_{trial} is the average time taken to complete a single trial. It is desirable to maximize N_{trials} for a given *Runtime* for maximizing wire length optimization. Therefore we need a low T_{trial} for a given *Runtime*. A straightforward strategy for achieving low T_{trial} could be to move cells in a random manner and accept the movement as valid if the wire length of the system reduces. This random position generation would have a low T_{trial}. However, this algorithm would still degrade the Total Improvement (TI) which can be expressed as follows: -

$$TI = N_{trials} \times A_{rate} \times \delta WL_{PA} \tag{1.2}$$

Where A_{rate} is the acceptance rate which is the number of acceptances per unit trial. δ WL_{PA} is the average wire length reduced per accepted movement. Random movement-based incremental placement would have a low A_{rate} and δ WL_{PA} because of the non-directed manner of solution generation. Therefore, there is a trade-off between *Runtime* and TI. Whereas, a Machine Learning(ML) model such as an ANN trained with appropriate data can achieve N_{trials} in order of what random algorithm would achieve given the simple MAC (multiply and accumulation) operations that the ANN needs to perform in a forward pass for predicting better positions of the cells. Also, high enough A_{rate} and δ WL_{PA} can be achieved only if the ANN model is trained with optimal data. Therefore, an ANN can provide a large wire length reduction for a given *Runtime* and hence a better trade-off.

1.3 Challenges

Major challenges involved in employing an ANN for incremental placement can be stated as follows: -

1. Training data needs to be realistic. It should correlate with the designs that the ANN model would see in the real world. Hence the training data has to be generated in a constrained random manner.
2. To ensure high A_{rate}, we need to ensure that the training data is optimal. Optimal data is which provides the smallest wire length positions as a solution for the example problems used to create the data-set.
3. ANN predictions and incremental placement algorithm using it should be fast enough to achieve high N_{trials}. Data pre-processing hence needs to be optimized.

Challenge 1 is addressed by controlling the randomness of the training data generation algorithm to direct it towards generating data close to real design cases. Challenge 2 is addressed by using a problem-directed data generation algorithm and a simple solution-directed data generation algorithm. To address challenge 3 we select the input features judiciously to avoid a large input vector size for the ANN. Additionally, we perform incremental placement by devising a fast and efficient method to optimally place a given cell in a small part of the design layout. Subsequently, we use this method over the entire layout to derive a placement solution that minimizes the cost for the given design.

1.4 Contribution of this work

This work is in continuation with the previous work on the same topic [13]. This work extends the previous work by performing the following tasks: -

1. Performing incremental placement in a grid of 10 $\times$ 10 to cover the complete design using a sliding window scan.

2. Generating problem solution examples to train the ANN model for its deployment in the incremental placement.

3. Modelling the connectivity of the design in a more accurate manner.

4. Making the ANN congestion aware by using the grid congestion as a feature in the ANN model.

5. Showing that using a different algorithm to generate additional training data improves the performance of the incremental placer significantly.

6. Achieving better performance than a random placement-based incremental placer by using a limited number of data samples as compared to the possible problem space.

Also, other improvements related to biasing the placement towards being more clustered and clustering while modeling the congestion have been incorporated.

1.5 Thesis Organization

We discuss the incremental placement problem formulation in chapter 2. The complete methodology for deriving the incremental placer is then described in chapter 3. It describes how we proceed to perform incremental placement by using the ANN which is used to infer the better positions of the cells in the design. In chapter 4, we describe the methodologies derived to obtain the training data for the ANN synthetically. We use two methods of data generation namely, Solution-Directed and Problem-Directed data generation approach. Once the data is generated, we need select and train an ANN model to predict cell positions according to the data generation heuristic. To train the model, we use a particular custom loss function and an ANN architecture which are described in chapter 5. Finally, we take the trained ANN model from the training algorithm and use it to predict better cell positions to perform the incremental placement which is explained in chapter 6. Finally, we explain the obtained results in chapter 7 which is followed by concluding the work in chapter 8.

Chapter 2

Incremental placement problem formulation

Incremental placement is performed post placement step. Once the global routing step is completed, we have a fair idea of how the wire length changes with the incremental changes in the design placement and congestion. Therefore the scenario present at the time of the incremental placement step is similar to the one present in the global routing step of the VLSI design.

For formulating the incremental placement scenario, we have used ICCAD 2020 problem as the reference to model the problem. Following are some important features of the design according to the modeling.

2.1 Global Grid (GG)

The given design layout is divided into grids which we refer to as Global Grid (GG). The GG can be visualized as a 3-D array of small cells which we refer to as GC. Third dimension of the matrix models the different metal layers present in the design. In Fig. 2.1, L1, L2 and so on show the third dimension. The GG has L_G Layers, R_G rows and C_G columns describing its dimensions.

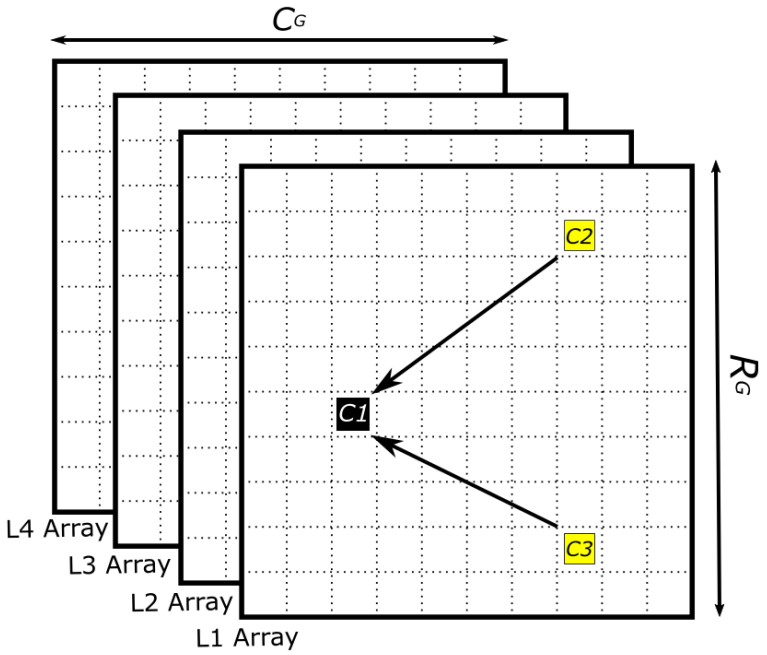

Figure 2.1: Arrays used for modelling the design

2.2 Cells (C_i)

Cell instances that are placed on design may lie inside any GC. We assign a unique index i to each cell in a design, and refer the cell as C_i. The position of a cell C_i is defined by the row and column of the Grid Cell (GC) that contains C_i, and is denoted by $row(C_i)$ and $col(C_i)$, respectively.

2.3 Nets (N_i)

Cells are connected to each other using connecting wires called nets. We assign a unique index i to each net in a design, and refer the net as N_i. To connect two cells lying at different location, nets are laid in the grid and the path it takes is called as a route.

Fig. 2.1 shows cells C_1, C_2 and C_3 connected using nets N_1 and N_2 as the cells lie in different GCs in layer L_1. Inside each GC only a limited number of routing-tracks are available. These routing-tracks define the path for the nets that cross the GC to

connect various GCs in the GG together. The number of routing-tracks available in the GC is the number of nets that can be routed using the GC. We term the number of routing-tracks present in a GC as supply (S_{GC}). The routing-tracks can be blocked by the nets routed through the GC or the cells that lie inside the GC. Number of routing-tracks blocked is termed as demands consumed inside the GC (D_{GC}). Number of tracks blocked by a cell is known as cell blockage (B_{C_i}). A net while passing through a GC consumes only one demand. Therefore, total number of demands created in the GC can be expressed as follows: -

$$D_{GC} = N_{GC} + \sum_{where C_i belongs to GC} B_{C_i} \tag{2.1}$$

The remaining number tracks which are not blocked by either a net or a cell inside the GC is termed as unused supply (U_{GC}). Therefore,

$$U_{GC} = S_{GC} - D_{GC} \tag{2.2}$$

Lower the value of U_{GC}, higher is the congestion inside a GC.

2.3.1 Wire length measurement

We measure wire length of a particular net by counting the number of GCs that a net crosses. We illustrate the wire length measurement using Fig. 2.2. Cells C_1, C_2 and C_3 are connected using a single net N_1 which is routed aas shown. According to our definition, since the route is crossing 13 GCs (excluding the C_1, C_2 & C_3 GC locations), the wire length is equal to 13 in this case.

2.3.2 Constraints for incremental placement

For a placement and routing to be valid, U_{GC} must always be greater than or equal to zero. If this condition is not satisfied, then the GC is said to be over-flown. While performing incremental placement, we need to satisfy the following constraints to avoid over-flow:

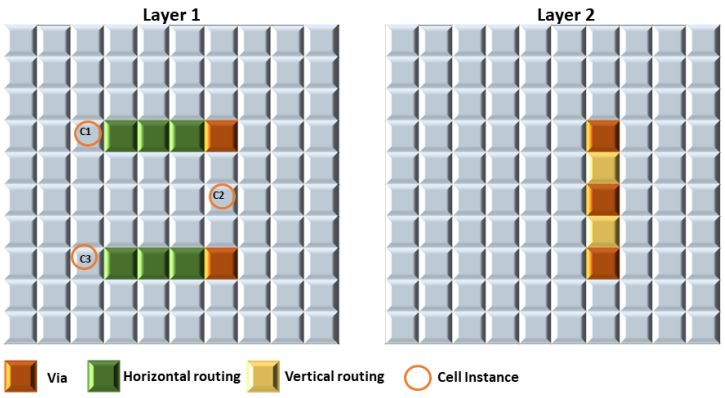

Figure 2.2: Routing and Wire length calculation

- Cell constraint : A cell C_i with blockage B_{C_i} can be placed in a GC only if U_{GC} is greater than or equal to the sum of B_{C_i} and the number of nets connected to the cell C_i.
- Net constraint: A net can be passed through a GC only if U_{GC} is greater than or equal to 1.

All the connections should be preserved while performing incremental placement. To ensure this, affected nets need to be re-routed after making incremental changes to the placement.

2.3.3 Objective

The objective of incremental placement is to make small changes in the placement of an already placed and routed design to decrease the overall wire length satisfying the above constraints. Small changes are made locally to not affect the placement completely and maintain the previously obtained placement solution.

Chapter 3

The methodology for deriving ANN based incremental placer

We derive an ANN based incremental placer and hence the quality of results depend upon the quality of training that the ANN is provided in the process of deriving the ANN based incremental placer. We shall now discuss what all steps we undertake for arriving at the final ANN based incremental placer.

In this work, we carry out incremental placement by processing a part of global grid (GG) at a time. We refer to a part of a GG as a slice. Similar to the GG, a slice consists of grid cells (GCs). However, a slice is significantly smaller than the GG and has less attributes/features than the corresponding GCs in the GG (these attributes are described in the next section). The small size and fewer features allow us to devise a fast and efficient incremental placement algorithm for the slice.

We apply the incremental placement algorithm of a slice to the entire layout using the sliding window scan technique.

We extract slices from the layout (highlighted as yellow grids) and perform incremental placement within the slice. Then, we extract another slice by including adjacent grids on the right and excluding grids on the extreme left. After we cover all the GCs horizontally, we move downward. We carry out this process until the entire layout is covered.

Fig. 3.1 illustrates the overall methodology. It consists of three main parts: training data generation, ANN model creation and validation, and incremental placement. In this work, we train an ANN to perform incremental placement for a slice. The ANN can predict a position for a cell that reduces the total wire length for nets within a slice.

In this work, we train the ANN model using data internally generated by our overall methodology. Due to small size of slices, we can quickly generate a large set of optimal problem-solution pairs for incremental placement. Data partitioning is done after the data generation to divide it into test (20%) and training (80%) data. Test data is used to validate the model after training. The training data is provided as an input to the ANN model training algorithm.

Model training algorithm trains the ANN model using the data generated in the previous step. The ANN architecture is decided experimentally. Training algorithm adjusts the weights of the ANN in such a way that it is able to predict position at which the cells should be placed for lesser wire length. Once trained, the model is tested using the test data obtained in the data partitioning step. Model is evaluated by predicting the outputs for test data reserved during data partitioning step. Predicted values are compared against the test data output expected values and the error is measured in terms of deviation from the test value. Once these errors are in acceptable range, the model is labelled as PASSED and can now be used for deploying in an incremental placement algorithm.

Finally, the trained model is plugged into the incremental placement algorithm. Incremental placer uses this ANN to predict optimal positions for cells in the slices created during the sliding window scan.

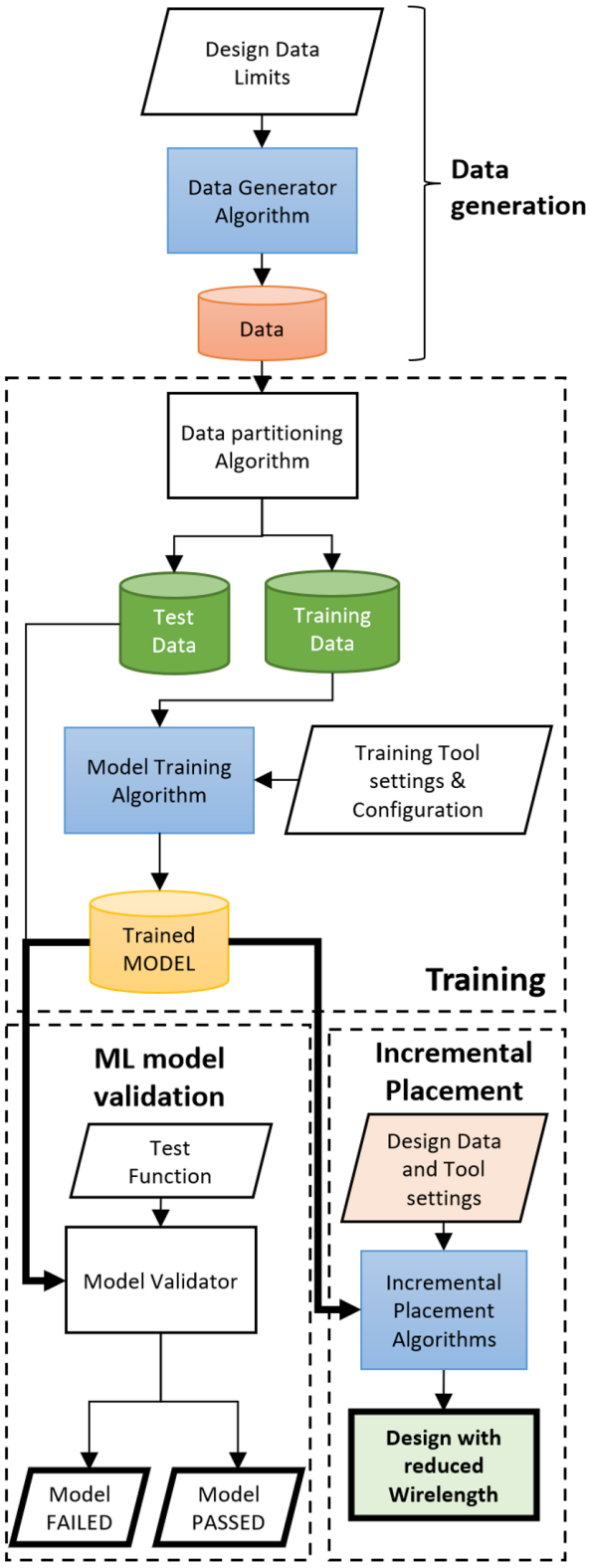

Figure 3.1: Overall Methodology

Chapter 4

Developing ANN model for predicting optimal locations

4.1 Data Generation

We train an ANN with relevant data such that it predicts positions for the cells in a slice that reduce the total wire length.

To train the ANN for predicting the cell positions resulting in lower wire length, we need a set of problem-solution pairs. Fig. 4.1 shows one of the example problem-solution pairs. Fig. 4.1(a) shows the initial position of C_1 along with connectivity in

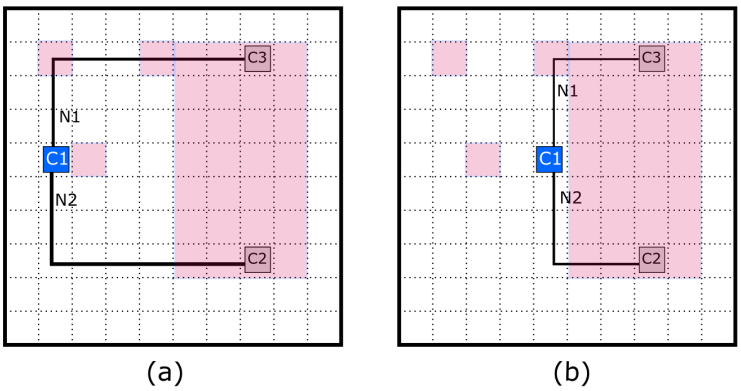

Figure 4.1: Example problem-solution pair

a slice. The initial wire length is 10 units (N_1 = 10 and N_2 = 10). Fig. 4.1(b) shows the slice after C_1 is moved. GCs highlighted in red are congested and therefore fail cell constraint. This incremental placement change reduces the wire length to 6 units (N_1 = 7 units and N_2 = 7 units). Hence, we can consider the above incremental change in the position of C_1 as a solution for the slice. For this problem-solution pair to be ideal, all other movements of C_1 that satisfy the cell and net constraints must produce a wire length of 6 units or more which is true about the example taken in Fig. 4.1 given the congestion profile.

4.1.1 Feature Selection

We observe that the following features affect the total wire length of the nets in a slice.

Connectivity

We represent the connectivity of the cells in a slice using a connectivity-matrix ($conMat$). Fig. 4.3 shows the connectivity matrix corresponding to design of Fig. 4.2

While forming the matrix $conMat$, if the cell C_i is connected to net N_j, then the index (i, j) is assigned a value of 1. Otherwise, it is assigned a 0. Therefore, two cells connected to the same net are said to be connected to each other. During data generation, we fix the maximum number of cells in a slice to $CellCount_{max}$ and a maximum number of nets to $NetCount$. Therefore, largest connectivity-matrix would be of size equal to $CellCount_{max} \times NetCount$.

Congestion

Fig. 4.4 shows how the wire length may change due to different congestion maps inside a slice. Nets can take only the paths on which unused supply U_{GC} are available and hence may need to take a detour due to congested GC lying in the shortest connecting path. Fig. 4.4(a) shows the routing of the nets when there is no fully congested GC in

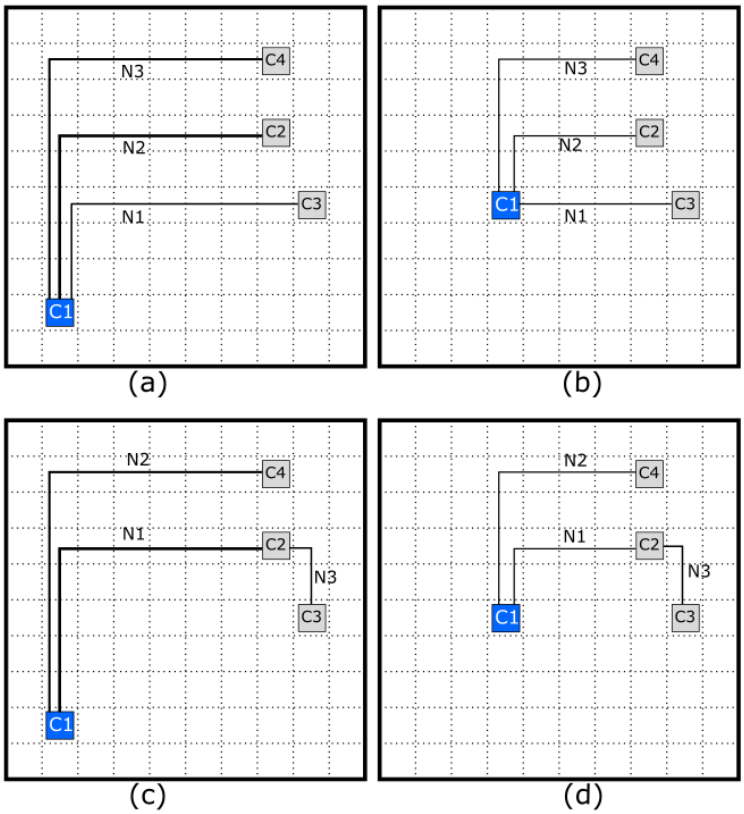

Figure 4.2: Effect of connectivity on wire length

the complete slice. This means that at-least one routing-track is available in each of the GCs. On the other hand, Fig. 4.4(b) shows how the nets change their route since the highlighted GCs are fully congested. This detour of net N_3 increases the wire length by two units. Therefore, we use U_{GC} to model congestion in the slice.

4.1.2 Complexity of data to be modelled

Tab. 4.1 lists the attributes of a slice and its components. We use these attributes as features for training the ANN model that predicts optimal location for cells in a slice.

Tab. 4.1 also shows the range and number of possible values for each of the attributes inside a slice considered in this work. We have taken $CellCount_{max}$=10 and $NetCount$=9 in this work. Additionally, we have taken slices of size 10×10. We have taken appropriate ranges for other attributes as shown in the table.

	N1	N2	N3
C1	1	1	1
C2	0	1	0
C3	1	0	0
C4	0	0	1

Figure 4.3: Connectivity matrix for Fig. 4.2(a-b)

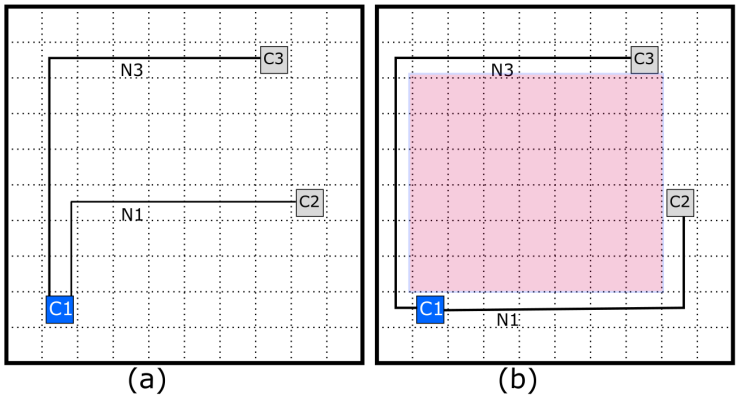

Figure 4.4: Effect of congestion on wire length

The above table suggests that we can generate $\approx 10^{243}$ slices by varying each attribute independently. Thus, if we want to exhaustively list and train the ANN for all possible slices, we will need $\approx 10^{243}$ examples. However, this is impractical. 10^6 We want to train the ANN model using around half a million examples so that the run time spent in training the model is reasonable. Therefore, we need to generate a small set of training examples that capture the placement and routing scenarios realistically. We achieve this by constraining the randomness of the data generation algorithm towards realistic scenarios, as explained below.

For generating the training data containing problem-solution pairs, we would need to generate random or constrained-random cell placement and connectivity. This is discussed in following sub-section. Tab. 4.2 lists all the attributes and their of the slice which would be used to explain the algorithms.

Table 4.1: Training Data Complexity

Range	Feature	#Possibilities
3 to 10	Number of cells, $CellCount$	7
1 to 10	Number of nets, $NetCount$	9
1 to 10	Row Position of each cell, $C_i.row$	10^{10}
1 to 10	Column Position of each cell, $C_i.col$	10^{10}
1 to 30	Each cell's blockage, B_{C_i}	30^{10}
1 to 1023	Connectivity matrix, $conMat$	1023^{10}
1 to 10	Index of cell to be moved, m	10
1 to 10	Position of C_m, ($C_m.row$, $C_m.col$)	10^2
1 to 30	Blockages of cell C_m (B_{C_m})	30
1 to 50	Unused supply, $uSupMat$	50^{100}

Table 4.2: Attributes of a slice

Attribute	**Abbreviation**	**Description**
Cells	C_i	Cell with index, i
Nets	N_i	Net with index, i
Cell blockage	B_{C_i}	Blockage of cell, C_i
Cell List	cL	Carries all the cells, C_i
Net List	nL	Carries all the nets, N_i
Connectivity Matrix	$conMat$	Net vs Cell connectivity
Unused supply matrix	$uSupMat$	U_{GC} at all GCs
Supply Matrix	$sMat$	S_{GC} at all GCs
Demand Matrix	$dMat$	D_{GC} at all GCs

4.1.3 Constrained random placement and connectivity generation

To generate a slice, first we generate constrained-random placement of cells, and then connect them systematically.

Constrained random placement generation

For a given list of cells in a slice (@cL), we first decide the cell to be moved (C_m). We place C_m randomly in a GC within the slice. Then, we place other cells of the slice randomly in the neighborhood of C_m. We use normal-distribution probability density function to

bias the placement of other cells closer to C_m. The probability of the row position x being generated for a cell is given as:

$$P(x) = \frac{1}{scF\sqrt{2\pi}} e^{-(x-C_m.row)^2/2\times scF^2} \tag{4.1}$$

where, scF is the scattering-factor, C_m.row is the row of the GC in which C_m lies. If scF is low, the location of a cell will be more probable to be closer to C_m. We use the above method of generating placement because we perform incremental placement after global routing. During the previous placement step, the connected cells would have been placed in the neighborhood and not randomly [14–17]. Therefore, this method of generating placement captures realistic scenario in a design. The Algo. 1 shows that pseudocode for placement generation in a slice.

Algorithm 1: PLACE_SLICE

Input: -
1. Slice: cell list @cL, dimensions (R_s, C_s)
3. Cell to be moved, C_m
4. Scattering factor, scF

Output: -
1. positions for cells in @cL

START:
$C_m.row$ = RAND(1, R_s)
$C_m.col$ = RAND (1, C_s)
foreach *cell in* @cL **do**
    if $cell \neq C_m$ **then**
        $cell.row$ = GAUSS_RAND($C_m.row$, scF, 1, R_s)
        $cell.col$ = GAUSS_RAND($C_m.col$, scF, 1, C_s)

Connectivity generation

We need to generate connectivity among cells in a slice randomly. However, when we are determining the location for a cell C_m that minimizes the total wire length, the connectivity of C_m with other cells is only relevant, as illustrated in Fig. 4.5. The slice contains cells $C_1 - C_6$ connected using nets $N_1 - N_4$. Assume that we move the cell C_1. As a result, the original connectivity (shown as solid nets) and wire length will change

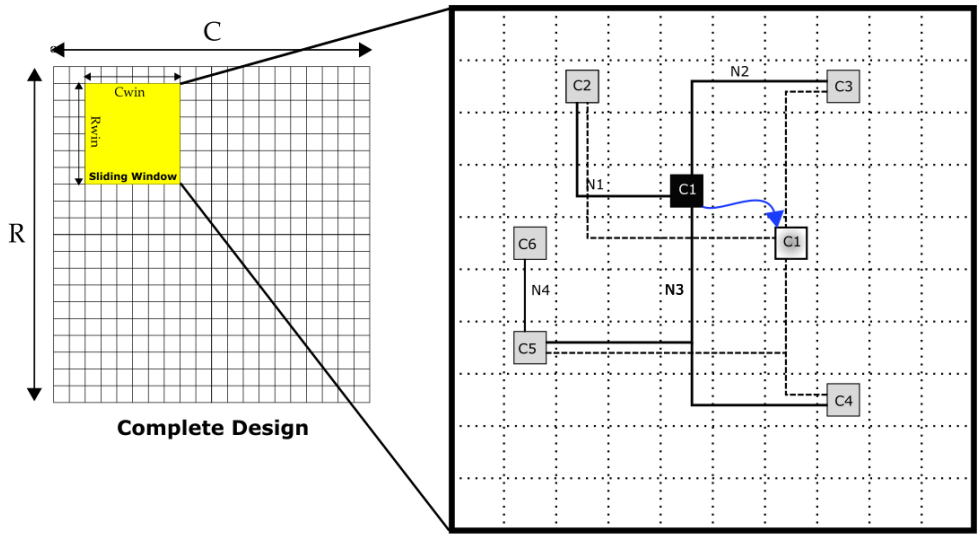

Figure 4.5: Scanning design and moving cells

(shown as dotted nets). However, the nets that are not connected to C_1 (such as N_4) do not have any change in the wire length. Hence, the nets connected to the cell to be moved C_m are only relevant. Hence, while generating connectivity for a slice, we first connect all the nets to the given cell C_m. Subsequently, we connect other cells to the nets randomly. The pseudo-code for connectivity generation is shown in Algo. 1.

Algorithm 2: CONN_SLICE

Input: -

1. slice with empty $connMat$
2. Number of nets ($NetCount$) and number of cells ($CellCount$)
3. C_m cell index (m)

Output: -

1. slice with filled $conMat$

```
for i = 1 to i = NetCount do
    for j = 1 to j = CellCount do
        if i = m then
            conMat[j][i] = 1
        else
            conMat[j][i] = RAND_SELECT(0, 1)
```

Th function CONN_SLICE takes maximum number of nets allowed for establishing the connectivity ($NetCount$), number of cells ($CellCount$), index of the cell to be moved (m) and an empty $conMat$ for the slice at input. We fill the elements of $conMat$

based on the above method of generating connections.

We now discuss the two methods that we use in this work to generate problem-solution pairs for training and testing of ANN-based incremental placer for a slice.

4.1.4 METHOD 1: Solution-directed data generation

In this method, we generate placement and connectivity, as described above. Then, we generate constrained-random congestion in the slice, followed by finding the minimum wire length position of a given cell in the slice.

Generating constrained-random congestion in a slice

First, we randomly generate cell blockages for each cell in the slice within the range [1, $BCMAX$]. Then, we randomly allocate unused routing supply to each GC within the range [1, $USMAX$]. Then, we randomly select a GC in the slice that has congestion. Further, we create $numTrialHS$ congestion hotspots around this GC. For creating a congestion in a GS, we allocate unused supply less than what is need for the cell C_m to be moved. We use normal-distribution probability density function shown in Eq. 4.1 to create clusters of congested GC. The pseudocode for generating constrained-random congestion is shown in Algo. 3.

Determining optimal location for a cell

Given a slice with constrained-random placement, connectivity, and congestion, we determine an optimal location of a given cell C_m. We temporarily place C_m at all the non-congested GCs one by one and determine the resulting wire length by performing virtual routing. The GC that achieves the minimum wire length is the required solution for the given problem.

We use congestion-aware maze routing (MR) algorithm to determine minimum wire length routes for a net [18]. However, due to constrained-random congestion in the

Algorithm 3: RANDOM_CONSTRAINTS

Input: -
1. slice: R_s, C_s, L_s, C_m
2. Number of trials for making hot-spot, $numTrialHS$
3. Congestion scattering factor, $congScF$
4. Maximum cell blockage ($BCMAX$) and maximum unused supply $USMAX$.

START:
// Initializing cell blockages
foreach *cell* C_i *in slice.@cL* **do**
  B_{C_i} = RAND (1, $BCMAX$)
// Initializing unused supply for slice
foreach GC *in slice* **do**
  $uSupMat[GC]$ = RAND (1, $USMAX$)
$demC_m = B_{C_m} + C_m.NetCount$
$rowHS$ = RAND (1, R_s)
$colHS$ = RAND (1, C_s)
// Creating congestion at the hot-spot seed
$uSupMat[rowHS][colHS]$ = RAND (0, $demC_m$ - 1)
for j = *1 to* $numTrialHS$ **do**
  r = GAUSS_RAND ($rowHS$, $congScF$, 1, R_s)
  c = GAUSS_RAND ($colHS$, $congScF$, 1, C_s)
  $uSupMat$[r][c] = RAND (0, $demC_m$ - 1)

grid, it is not always possible to route a net for an arbitrary position of C_m. Hence, wire length cannot be determined for some positions of C_m and we discard those positions [19].

For C_m connected to multiple nets, we perform routing of each net sequentially. However, the wire length will depend on the chosen sequence of routing because routing a net changes the congestion profile for a slice. Further, routing for a net can fail because of the routing resources consumed by the previously routed nets. We follow a one-level rip-and-reroute strategy to avoid sequence of net that yields routing failure.

Wire lengths obtained by routing the nets after placing C_m at all locations temporarily are compared to find locations (GCs) which result in minimum wire length. These locations are regarded as solution to the constrained random problem generated in the previous steps.

The 5×5 grid shown in Fig. 4.6 illustrates the process of determining the optimal location for the cell C_1. The cells C_2 and C_3 are connected to C_1 using net N_1.

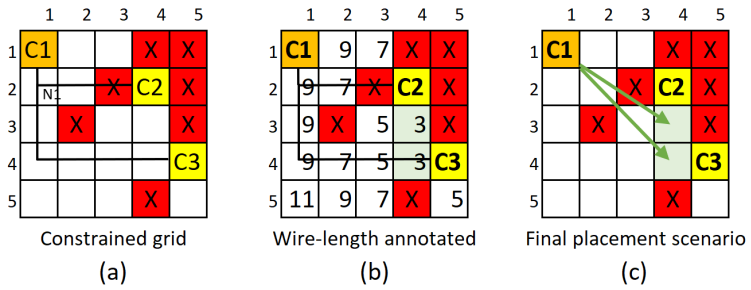

Figure 4.6: Solution directed data generation

The cell constraint fails for the GCs shown in red. First, we determine the total wire length by placing C_1 at each GC and performing congestion-aware virtual routing (wire length annotated in Fig. 4.6(b)). The GC at (4, 4) and (3, 4) yields the minimum wire length. Hence, these are the optimal positions for C_1 (as shown in Fig. 4.6(c)). We will treat the given problem and these two optimal locations as two data points for training and testing of ANN-based incremental placer for a slice.

Limitation of solution-directed approach

Despite performing an exhaustive search over the slice, the above method can produce locations that yield sub-optimal wire length. The optimality of solution depends upon the chosen sequence of nets for carrying out virtual routing.

To overcome this limitation, we propose another data generation algorithm (problem-directed) which ensures optimal problem-solution pairs while still using inherently sequential MR algorithm for routing the nets.

4.1.5 METHOD 2: Problem-directed data generation

In this method, we first create a constrained-random placement and connectivity in a slice. Further, we assume that the current position of a given cell C_m at $GC(i, j)$ is optimal (yields minimum wire length). Subsequently, we determine another position of C_m at $GC(x, y)$ and adjust the congestion profile in the slice such that $GC(x, y)$ is sub-optimal. Thus, we obtain a placement problem with C_m at $GC(x, y)$ for which moving it

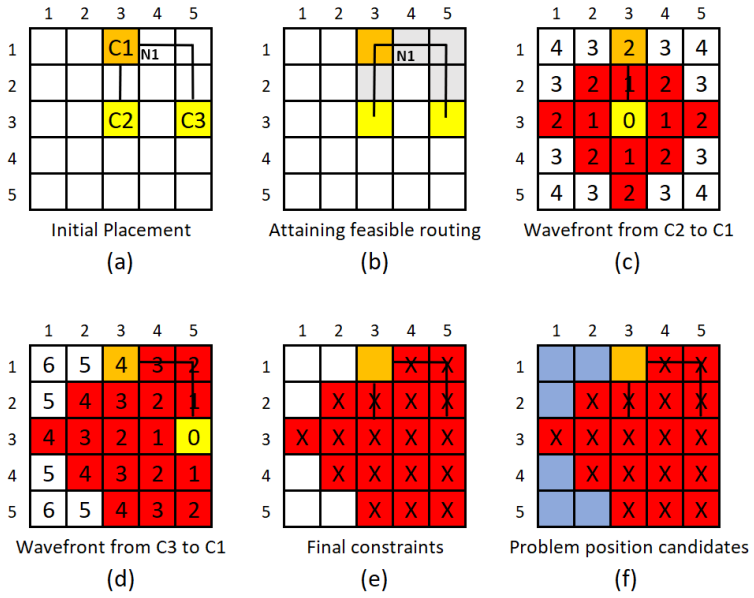

Figure 4.7: Example to understand finding constraints function

to $GC(i, j)$ is an optimal solution. Since we start with the solution $GC(i, j)$ and determine the problem $GC(x, y)$, we refer to this approach as problem-directed. It is easier to ensure that the position $GC(i, j)$ is optimal because we can manipulate the congestion profile such that all the superior solutions are ruled out for the generated problem.

We illustrate the above process in Fig. 4.7. Firstly, we generate random-constrained placement and connections using Algo. 1 and 2. There are three cells C_1, C_2 and C_3 all connected to each other using net N_1. Assume that we consider cell C_1 for movement, i.e., $C_m = C_1$. We consider the current position of C_1, i.e. $GC(3, 1)$ as the optimal solution. Subsequently, we formulate a placement problem for C_1 such that its optimal location is $GC(3, 1)$.

Firstly, we determine U_{GC} constraints such that it is feasible to route nets without detour when C_1 is placed at $GC(3, 1)$. We ensure that the net constraints and cell constraints are honored for this configuration of the slice. Fig. 4.7(b) shows the GCs that N_1 passes and we need to ensure net constraint on these GCs. Next, we add U_{GC} constraints in the slice such that the superior solutions become infeasible for routing (we ensure this by making the cell constraints fail for these GCs). To achieve this we propagate wavefront from each connected cell (C_2 and C_3) towards C_1 (as shown in Fig. 4.7(c) and (d)) using maze routing. The U_{GC} for the set of GCs that yield superior

solution than $GC(3,1)$ (shown in Fig. 4.7(e)) are adjusted such that routing becomes infeasible for these positions of C_1. Subsequently, we choose the locations of C_1 from the set of locations (shown as blue cells in Fig. 4.7(f)) for which $GC(3,1)$ will be an optimal solution. Moreover, the location $GC(3,1)$ will be optimal because we ensure that a direct path (without any detour) exists for nets between C_1 and other cells.

For creating data samples for our data-set, the blue GCs in Fig. 4.7(f) become $GC(x,y)$ position candidates for the single optimal solution $GC(i,j)$ highlighted in orange.

4.2 Training the model

We generate training and test data using both solution-directed and problem-directed approaches. The solution-directed approach covers a wide range of problem space since we allow position, connectivity, and congestion to vary in a wide range. However, the solutions to these problems may be sub-optimal in some cases. The problem-directed approach guarantees an optimal solution by tightly constraining the congestion profile. However, it cannot capture the problem space widely because of the restriction in the congestion profile. Hence, both these methods have their pros and cons, and we use a combined data set for training and testing the ANN model. We have generated a total of 500K problem-solution pairs. Further, we use 80% of them for training and 20% for testing the model.

We train an ANN model to predict an optimal location for a cell C_m in a slice with given cell positions, connectivity, and congestion profile.

ANN consists of multiple hidden layers between the input layer and the output layer which define the architecture of the model. We select an architecture that achieves an acceptable accuracy in predicting the optimal location for a cell C_m.

The training algorithm needs to calculate loss (error) by a given loss function for the predicted position for adjusting the weights of the ANN. Assume that $GC(row_{true}, col_{true})$ denotes the true position (expected to be predicted) and the ANN produces $GC(row_{pred}, col_{pred})$ as the predicted position. In this work, we have defined a custom loss function based on two measures:

1. Position Loss: We define the position loss POS_{loss} as the half perimeter wire length (HPWL) between these two positions:

$$POS_{loss} = |row_{pred} - row_{true}| + |col_{pred} - col_{true}| \tag{4.2}$$

   The position loss denotes how far the predicted position is from the true expected position.

2. Wire length Loss: We calculate the total wire lengths WL_{true} and WL_{pred} by carrying out virtual congestion-aware maze routing for the true and the predicted positions, respectively. We define wire length based loss WL_{loss} as:

$$WL_{loss} = \frac{WL_{pred} - WL_{true}}{WL\_CONST} \tag{4.3}$$

   When the ANN predicts a position where the cell-constraint or the net-constraint are violated, we take the WL_{pred} as infinity (a very high value). WL_CONST is a scaling factor.

We combine the above loss function as follows:

$$Loss_{total} = (1 - \alpha) \times POS_{loss} + \alpha \times WL_{loss} \tag{4.4}$$

where, $0 < \alpha < 1$. We have used $\alpha = 0.3$ in our work. By combining the two loss functions, we ensure that the model is trained for the minimum wire length, rather than the given position. This formulation is helpful in situations in which there are multiple locations (GCs) that yield optimal wire length.

For selecting the ANN architecture, we try different architectures and evaluate them after training them for same number of epochs with similar loss-function. Tab. 4.3 shows the mean-absolute position error ($MAPE$). $MAPE$ can be expressed in terms of row and column error for a batch of training data samples.

$$MAPE = \sqrt{MAE_{row}^2 + MAE_{col}^2} \tag{4.5}$$

Where, MAE_{row} and MAE_{col} are mean absolute row and column errors for a batch of training data samples. for different ANN architectures.

Table 4.3: Architecture comparison for model evaluation ($\alpha = 0.3$)

Architecture	Error (MAPE)
248, 124, 2	7.58
248, 248, 248, 248, 2	7.58
248, 500, 248, 124, 2	0.566
248, 500, 500, 248, 248, 124, 2	0.5
300, 600, 300, 150, 100, 50, 25, 15, 5, 2	0.58

Starting from a simple ANN architecture, we keep modifying it until we arrive at an architecture which has an acceptable error. We change the number of layers, the number of nodes, and the sizes of individual layers experimentally to arrive at architecture highlighted in green in Tab. 4.3. Changing this architecture further reduces the accuracy and hence we use this architecture in the ANN-based incremental placement methodology.

4.3 Incremental Placement Algorithm using ANN model

As we have created a trained ANN model that can predict better placement for a cell instance for a slice of dimension $R_s \times C_s$. To perform incremental placement on large dimension design, we scan the design to extract features of the design into a slice of small dimension and perform incremental placement in the slice.

Algorithm 4: Incremental Placer

Input:
GG with all design info.
Output:
GG with incremental placement done.
START:
cond = True;
while *cond* = *True* **do**
    $(slice, C_m) = GET_NEXT_SLICE (*GG*);
    $\$GC(r_{pred}, c_{pred})$ = ANN (slice, C_m);
    if *IMPROVE* (C_m, $\$GC(r_{pred}, c_{pred})$, *GG*) **then**
        COMMIT_MOVE (C_m, $\$GC(r_{pred}, c_{pred})$, *GG*);
    cond = CHECK_COND (*GG*);

Algo. 4 shows the tasks that the ML based incremental placer performs. Input to this design are derived from the ICCAD input design file which is the complete GG.

We perform a sliding window scan of the complete large design as illustrated in Fig. 4.5. This sliding window extracts a slice at each step of the sliding window scan (using GET_NEXT_SLICE). The sliding window scan continues until the condition, *cond* is not false. This condition is checked at the end of each step of sliding window scan (CHECK_COND) which returns false if any of the following conditions fail.

- I: Sliding window reaches the end of the dimensions of GG.
- II: Time limit put over the sliding window scan reached.

On the slice obtained at each step, we perform incremental placement by predicting the better position $GC(r_{pred}, c_{pred})$ for C_m from the trained ANN.

This prediction is checked for following conditions by IMPROVE function:

1. Cell and net constraints are satisfied after moving C_m to predicted GC.
2. Wire length of system is lesser that earlier when C_m is placed at $GC(r_{pred}, c_{pred})$.

IMPROVE returns true only of both these conditions are satisfied once C_m is moved. Once checked, the movement is accepted as valid (by COMMIT function) and C_m is moved to predicted position in the GG and C_m enters the list of cells ($@MovedCells$) that are moved during sliding window scan based incremental placement.

Chapter 5

Results

First, we train two ANN models using a data-set obtained from the problem-directed mode and solution-directed mode data generation algorithm with an acceptable error. Also, we create a data set by mixing data from both algorithms to train another model.

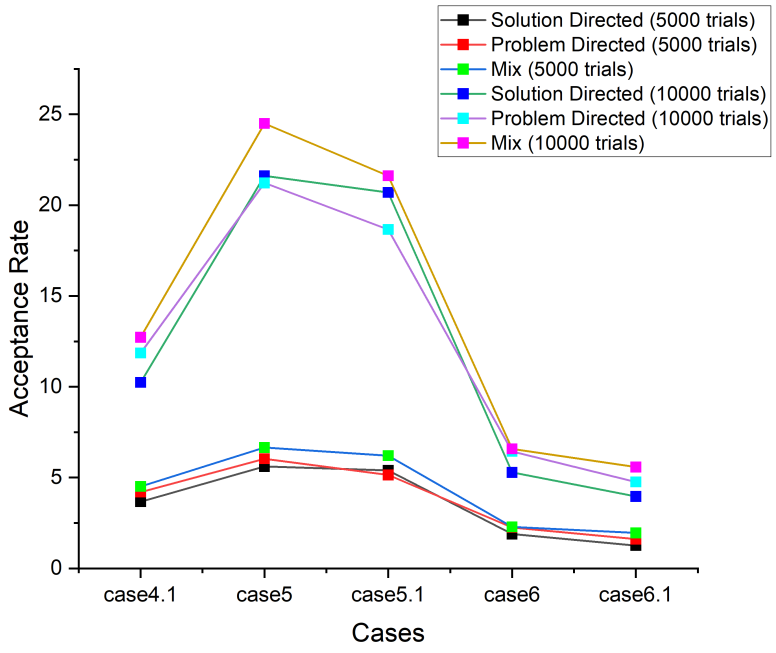

Figure 5.1: Acceptance rate comparison for ANN trained with different data generated from different algorithms

All three models are plugged into the incremental placement algorithm for comparison. Tab. 5.1 shows the performance of incremental placer based upon three different ANN models. We have used ICCAD 2020 designs for bench-marking and comparing the results. All experiments have been carried out on a 2.2 GHz Intel processor with Linux OS and 32 GB RAM. The figure 5.1 compares the A_{rate} for models trained with different data sets. The number of trials was fixed to 10000 and 5000 in two different runs, and we observed that the ANN-based upon Mix data evaluates to be best in both cases. We obtain a 15% average improvement in acceptance rate after incremental placement by using ANN trained with mixed data (P + S) compared to the ANN trained with a single (P or S) data set.

Also, we have defined 3 hours as the upper run time limit for incremental placement while noting the results. Since the mixed data-based ANN model evaluates to be the best. We use this model to further benchmark the results against a random position generation-based incremental placement algorithm. In this random positions-based algorithm, the ANN model is replaced by the random position generator which moves the C_m to random positions. The random position is checked for wire length reduction and is accepted as a solution if the wire length reduces. Also, the new position is checked for cell and net constraints before accepting it as a solution.

Tab. 5.1 shows the overall results with respect to TI expressed in terms of equation 1.2. Wire length reduced is termed as Score (δWL) which is also equal to total improvement (TI). $N_{trials_{acc}}$ is the number of trials that resulted in a movement actually being accepted. The average A_{rate} of the ANN-based algorithm is much high and N_{trials} are of the same order when compared to the random placement-based incremental placement algorithm. This results in higher average total improvement TI ($A_{rate} \times \delta WL_{PA} \times N_{trials}$) for the ANN-based algorithm. ANN-based algorithm's average TI is around 3 times the random placement-based incremental placement algorithm's average TI. A high δWL_{PA} indicates that the quality of the solutions provided by the ANN-based incremental placer is much better than the random placement-based incremental placer.

Hence, as expected from the ANN model, it is able to achieve a number of trials in a given run-time (N_{trials}) of the same order as the inherently fast random position generation-based algorithm. Also, the acceptance rate (A_{rate}) is 5 times the random placer and the wire length reduced per accepted movement (δWL_{PA}) is 20% higher for

Table 5.1: Machine learning based vs. Random Position based Incremental placement comparison

	Mixed data based ANN						Random position based					
Cases	Score (δWL)	N_{trials}	$N_{trials_{acc}}$	A_{rate}	δWL_{PA}	TI (A_{rate} * δWL_{PA} * $N_{trial}s$)	Score (δWL)	N_{trials}	$N_{trials_{acc}}$	A_{rate}	δWL_{PA}	TI (A_{rate} * δWL_{PA} * N_{trials})
case1	33	8	6	75	5.5	33	6	8	1	13	6	6
case2	10	6	2	33	5	10	0	6	0	0	0	0
case3	2347	6177	663	11	3.54	2347	840	5490	239	4	3.51	840
case3.1	1310	2822	351	12	3.73	1310	401	2775	118	4	3.4	401
case4	10411	18267	1327	7	7.85	10411	5344	18097	794	4	6.73	5344
case4.1	11303	18159	1412	8	8	11303	5697	18363	747	4	7.63	5697
case5	9154	15718	1914	12	4.78	9154	2638	14489	508	4	5.19	2638
case5.1	8003	15697	1719	11	4.66	8003	2187	14465	470	3	4.65	2187
case6	4678	14524	671	5	6.97	4678	1329	14171	244	2	5.45	1329
case6.1	3645	14366	542	4	6.73	3645	891	14326	169	1	5.27	891
	Average Score = 5089.4			Avg A_{rate} = 17.8		Avg = 5089.4	Average Score = 1933.3			Avg A_{rate}= 3.9		Avg = 1933.3

the proposed incremental placement methodology.

This provides a better run time and QOR trade-off for the incremental placement problem. The proposed algorithm/methodology can therefore provide a better solution in terms of wire length in the same amount of run-time as compared to random placement-based incremental placer.

Chapter 6

Conclusions and Future Scope

The ANN-based approach is more directed/converging as compared to the random position-based approach for incremental placement. Following are the conclusions that we can draw from the obtained results.

1. We generate the training data for the ANN using two different algorithms which have their own pros and cons. Both these algorithms are intentionally biased towards realistic designs and an algorithm that is a combination of both models the real case scenario more effectively.
2. Data generation is done independently of any pre-existing algorithms. This allows us to generate any amount of data for training and testing the ANN. We can tweak various variables of our algorithm to obtain data sets of different nature. This makes this approach scalable.
3. This ANN-based incremental placer results in multiple times better wire length reduction per movement and acceptance rate when compared to a random placer in a given run time.

The major advantage of this methodology is that its run time is comparable to random position-based incremental placer but with much better optimization capability.

We see a scope of improvement/expansion in various aspects of the proposed

ANN-based incremental placement developed. Also, there is a scope of comparison against some pre-existing conventional incremental placement algorithms. Following is the future scope for the derived placer: -

1. Comparing the results for the ML-based incremental placer against the total improvement that would be provided by the incremental placer working with conventional algorithms. The conventional algorithm is expected to arrive at a better acceptance rate but the number of trials per unit time is expected to be lower. Therefore, the overall TI is expected to be lower than the ML-based incremental placer.

2. Embedding this incremental placer into open source tools that perform the complete RTL to GDS flow.

3. Using CNN (Convolutional Neural Network) instead of ANN for deriving the incremental placer. Using CNN is intuitive since we are working with a 2-D grid in our problem formulation which can be handled in a better way by the CNN which inherently works well with 2-D images and is commonly employed for the same.

4. We can partition the design into several parts and use *multi-threading* to run the sliding window scan on each partition separately on each of the available CPU cores. This would further increase the number of trials per unit time and hence the overall QOR.

Bibliography

[1] N. Viswanathan, C. Alpert, C. Sze, Z. Li, and Y. Wei, "ICCAD-2012 CAD contest in design hierarchy aware routability-driven placement and benchmark suite," *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, pp. 345–348, 2012.

[2] A. Kennings and K. Vorwerk, "Force-directed methods for generic placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 2076–2087, 2006.

[3] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf placement and routing package," *IEEE Journal of Solid-State Circuits*, vol. 20, pp. 510–522, Apr. 1985.

[4] Y.-W. Chang, Z.-W. Jiang, and T.-C. Chen, "Essential Issues in Analytical Placement Algorithms," *IPSJ Transactions on System LSI Design Methodology*, vol. 2, pp. 145–166, 2009.

[5] H. Ren, D. Z. Pan, C. J. Alpert, and P. Villarrubia, "Diffusion-based placement migration," *Proceedings - Design Automation Conference*, pp. 515–520, 2005.

[6] U. Brenner, "VLSI legalization with minimum perturbation by iterative augmentation," *Proceedings -Design, Automation and Test in Europe, DATE*, pp. 1385–1390, 2012.

[7] H. F. Tsao, P. Y. Chou, S. L. Huang, Y. W. Chang, M. P. H. Lin, D. P. Chen, and D. Liu, "A corner stitching compliant B-tree representation and its applications to analog placement," *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, pp. 507–511, 2011.

[8] L. Xiao and E. F. Young, "Analog placement with common centroid and 1-D symmetry constraints," *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, pp. 353–360, 2009.

[9] M. K. Hsu and Y. W. Chang, "Unified analytical global placement for large-scale mixed-size circuit designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 9, pp. 1366–1378, 2012.

[10] A. B. Kahng and S. Reda, "Wirelength minimization for min-cut placements via placement feedback," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 7, pp. 1301–1312, 2006.

[11] A. R. Agnihotri, S. Ono, C. Li, M. C. Yildiz, A. Khatkhate, C. K. Koh, and P. H. Madden, "Mixed block placement via fractional cut recursive bisection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 5, pp. 748–760, 2005.

[12] P. Maidee, C. Ababei, and K. Bazargan, "Timing-driven partitioning-based placement for island style FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 3, pp. 395–406, 2005.

[13] S. Saurabh, P. Jain, M. Agarwal, and O. S. Ram, "Applications of machine learning in vlsi design," in *VLSI and Hardware Implementations Using Modern Machine Learning Methods*, pp. 125–139, CRC Press, 2022.

[14] M. Fogaça, A. B. Kahng, R. Reis, and L. Wang, "Finding placement-relevant clusters with fast modularity-based clustering," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, (Tokyo Japan), pp. 569–576, ACM, Jan. 2019.

[15] A. B. Kahng and R. Sharma, "Studies of clustering objectives and heuristics for improved standard-cell placement," 1997.

[16] A. Caldwell, A. Kahng, and I. Markov, "Improved algorithms for hypergraph bipartitioning," in *Proceedings 2000. Design Automation Conference. (IEEE Cat. No.00CH37106)*, pp. 661–666, Jan 2000.

[17] H. Chen, C.-K. Cheng, N. chi Chou, and A. Kahng, "An algebraic multigrid solver for analytical placement without layout based clustering," in *Proceedings 2003. Design Automation Conference (IEEE Cat. No.03CH37451)*, pp. 794–799, 2003.

[18] F. Rubin, "The lee path connection algorithm," *IEEE Transactions on Computers*, vol. C-23, no. 9, pp. 907–914, 1974.

[19] A. Deza, C. Dickson, T. Terlaky, A. Vannelli, and H. Zhang, "Global routing in vlsi design algorithms, theory, and computational practice," *JCMCC-Journal of Combinatorial Mathematicsand Combinatorial Computing*, vol. 80, p. 71, 2012.

Appendix A

Solution Directed Data generation

Algorithm 5: Data Generation - Solution Directed

Input: 1. *slice*, maximum Cells ($numCells_{max}$) and, maximum nets ($numNets_{max}$)
2. Number of samples to be generated $maxSamples$
3. List of scattering factors @$scFactorList$
Output: Training Data in CSV File named dataFile
START: INITIALIZE K_s, $numSamples$, $scFactorIdx$, $numscFactors$
while $numSamples < maxSamples$ **do**
  while $K_{win} > 2$ **do**
    while $scFactorIdx < numscFactors$ **do**
      $numNets_{win} = numNets_{win_{max}}$;
      while $numNets_{win} > 2$ **do**
        $connMat[numCells_{max}][numNets_{max}] = 0$
        $slice.@cellList$ = PLACE_SLICE ($slice.@cellList$, R_s, C_s, $cell_m$, $scFactorList[scFactorIdx]$);
        $connMat$ = CONN_SLICE ($connMat$, $numNets$, $numCells$, m)
        RANDOM_CONSTRAINTS ($slice$)
        (@$rowMinWL$, @$colMinWL$) = MIN_WL_POS($design$);
        @$minWLPos$ = (@$rowMinWL$, @$colMinWL$)
        WRITE_DATA ($slice$, @$minWLPos$)
        $numSamples$ += SIZE (@$minWLPos$)
        $numNets_{win}$–;
      $scFactorIdx$++;
    K_{win}–;

Appendix B

Problem Directed Data generation

Algorithm 6: Data Generation - Problem Directed

Input:-

1. Grid G of size R_{win} rows, C_{win} columns, L_{win} layers, Max Number of Cells $K_{max_{win}}$ and, Maximum number of nets $numNets_{win_{max}}$
2. Number of samples to be generated $maxSamples$
3. List of scattering factors $scFactorList$

Output: Training Data in CSV File named dataFile

INITIALIZE: K_{win}, $numSamples$, $scFactorIdx$, $numscFactors$

while $numSamples < maxSamples$ **do**
  while $K_{win} > 2$ **do**
    while $scFactorIdx < numscFactors$ **do**
      $numNets_{win} = numNets_{max_{win}}$;
      while $numNets > 2$ **do**
        $design$ = PLACE_SLICE ($design$, K_{win}, $numNets_{win}$, $scFactorList[scFactorIdx]$);
        $connMat$ = CONN_SLICE ($connMat$, $numNets$, $numCells$);
        $cstrMat$ = APPLY_CSTR ($design$);
        ASSIGN_DEM ($slice$, $cstrMat$)
        GENERATE_PLACEMENT_SCENARIOS ($design$, $dataFile$);
        $numNets_{win}$- -;
      $scFactorIdx$++;
    K_{win}−;
  $numSamples$++;

Brief Description of Author

Gagandeep holds a degree of B.Tech in Electrical and Electronics Engineering from Maharaja Surajmal Institute of Technology - Affiliated to Guru Gobind Singh Indraprastha University in the year 2017. He was awarded a gold medal for academic performance in B.Tech. He completed his M.Tech. in VLSI and Embedded Systems design from Indraprastha Institute of Information Technology, Delhi in 2022. His interests lie in the application of ML in VLSI Design and, Analog and Mixed signal circuit design. He is working as an Engineer Trainee in Synopsys India Pvt. Ltd., Noida. He can be reached at gagandeep20320@iiitd.ac.in or dipgagn@gmail.com.