MachineFlow: A Web Tool to Adaptively Select EDA Tool Parameters for Digital ICs Using Statistical Learning

by

Ariq Emtenan

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2016

Abstract

Digital integrated circuits (ICs) are the driving force behind computing, communication and entertainment in today's world. More powerful and energy efficient ICs continue to be designed and built by the semiconductor industry to meet current demands. In addition, complexity in Electronic Design Automation(EDA) software continues to grow to keep up with the design and technological advances in ICs. Design parameters that control various aspects of EDA software, such as synthesis, placement and routing settings, can affect the performance of ICs, and current standard cell logic synthesis and physical design flow tools consist of hundreds of such parameters to be specified by the designer. Discovering the optimal value for each parameter can result in significant performance and power efficiency gains. In this thesis, we develop a proof of concept web tool that uses open source EDA software to generate datasets for a large number of circuits and uses the data to build predictive models using statistical learning techniques. We use these models to select design parameters for the VLSI physical design flow floorplanning and placement stages that minimize the total wirelength of the integrated circuit. Using empirical evaluation on three real-world test circuits, we show a 15x-35x reduction in time spent to discover predicted values of one of the studied parameters that reduces total wirelength with statistical significance compared to a brute force exploration of the design space, with an error rate below 5%.

The key contributions of this thesis are:

- Development of a web based tool to perform VLSI physical design flow of a standard cell based integrated circuit, as well as to generate and record datapoints at various stages of the flow.
- Development and investigation of predictive models through the web based tool, using the recorded datapoints and utilizing statistical learning techniques, to predict VLSI physical design flow floorplanning and placement tool parameters that minimize total wirelength in a particular integrated circuit.

Acknowledgements

I would like to thank Professor Siddharth Garg, my thesis advisor and mentor at the University of Waterloo. Professor Garg has provided me with the wonderful opportunity to participate at exciting research at the University of Waterloo and has been a source of encouragement, support and motivation throughout my studies. Professor Garg introduced me to the world of electronic design automation which has been the central theme of my research, as well as the tools and software used in this field. He encouraged me to develop my own ideas for my thesis while ensuring I was on the right track. I would like to thank him for the time, effort and energy he has spent on my research work.

I would also like to thank my co-supervisor Professor Hiren Patel. Professor Patel has also been a source of support to me during my studies and research work at the University of Waterloo. The weekly student paper presentation sessions organized by Professor Patel introduced me to contemporary research topics and greatly expanded my horizons.

I would also like to express sincere gratitude to Professor Peter Levine and Professor Mahesh Tripunitara. Professor Levine has helped me greatly by providing early feedback after reading my thesis and by making concrete suggestions that improve the thesis. I gained a better understanding of my research while thinking about and addressing Professor Levine's comments on my thesis.

Professor Mahesh Tripunitara taught me practical software engineering skills and techniques in my first semester at the University of Waterloo, which have been extremely valuable in many software projects I've worked on since, including the MachineFlow software project. I would also like to thank Professor Tripunitara for providing valuable feedback to me during the seminar and agreeing to be a reader for my thesis.

I would also like to thank the Electrical and Computer Engineering Department and Faculty at the University of Waterloo for allowing me to study and work in an environment that fosters creativity.

Dedication

I dedicate this thesis to my father Md. Abdul Kafi and my mother Mrs. Shanaj Kafi for their love, support and the sacrifices they've made to educate me and my sister Anika.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The integrated circuit industry, in which digital integrated circuits have the largest marketshare, has been one of the core drivers in the current information revolution. Affordable and powerful electronic devices such as smartphones, tablets, laptops, as well as supporting backend infrastructure, such as data-centers and telecommunications networks have transformed the way humankind works, communicates and lives in today's world. Digital integrated circuits, such as microprocessors, memory chips, wireless and wireline tranceivers, sensors, displays and other ASICs form the core components in the devices driving this revolution.

Performance and power efficiency in digital integrated circuits have continued to improve to keep up with the demands in today's world, governed by Moore's law and Dennard scaling respectively. A dizzying array of design and technological improvements have kept both these trends alive for over 40 years. State of the art digital integrated circuits include the Intel Xeon ® E5-2600 v3 processor which packs 5.56B transistors on a 31.9mm x 20.8mm die providing 18 dual threaded processor cores at upto 3.8GHz frequency[3]; and the Intel 6th generation Core processor with feature size of 14 nm built using tri-gate technology, consuming 90mW of power on idle[11].

Circuit designers rely on Electronic Design Automation software such as Synopsys Design Compiler, Synopsys IC Compiler, Cadence Encounter etc to create new digital integrated circuits. As the semiconductor industry evolves to accommodate more transistors at smaller geometries, EDA tools need to add an increasing number features to support these changes. For example, IC Compiler II, the current Place and Route solution from Synopsys, allows designers to add trigate transistors known as FinFETs to their designs. Support for FinFETs adds a new power optimization metric to enable designers to balance power goals[36]. This adds to existing design parameters that designers already need to manually tune during the design process; for example, during the floorplanning stage using Cadence Encounter tool, designers can set values for the desired aspect ratio, core utilization, core margins, row spacing and so on[8]. Some of these parameters have default values, but the designer can get better results by using a value different from the default based on the circuit under design. For example, the core utilization metric which determines the density of standard cells in the core has a default value of 70% to make room for in place optimization and clock tree synthesis cells - however for more complex designs the designer can get better results by reducing the default utilization value.[8].

Support for new features results in an increase in the complexity of EDA Tools, and designers have to tune an increasing number of design parameters. This takes a toll on engineers working in design roles[35]. Over the years, while IC Complexity has grown by 58 percent, designer productivity has grown by 21 percent, resulting in the phenomenon known as the design productivity gap[1].

The other effect of increasing complexity in the integrated circuit design process is the widening supply gap between IC design and software engineers[19]. Many students in Engineering at University look for a quicker return on their education and choose a career in the software industry instead of devoting more time in post-graduate education needed for an IC designer position. Building better tools that allow students to retain their interest in circuit design in their undergraduate education is a challenge that needs to be solved to combat this gap.

The objective of this thesis is two-fold. The first objective is to create an intuitive web based interface on top of open source EDA Software to perform the ASIC design flow, using a cloud computing paradigm to perform the design flow on a server through actions by users on a web browser. The second objective is to incorporate features in this web-based tool that uses statistical learning models to simplify certain aspects of the digital integrated circuit design process, in particular, automating the selection of EDA tool parameters that minimize the total wire-length of the integrated circuit. We study various predictive models to evaluate the best approach, and explore the effectiveness of these models in selecting design parameters for the VLSI physical design flow floorplanning and placement stages.

In summary, through this thesis, we demonstrate a way to address the EDA tool complexity issue, helping improve designer productivity and making the EDA tool more approachable for undergraduate students interested in integrated circuit design by utilizing a familiar and intuitive web based solution.

Chapter 2

Background

Digital integrated circuits can be designed using a number of implementation strategies. Custom circuit methodology involves building the circuit topology and performing the physical design without using design automation tools. Custom circuits can achieve the best performance, but is expensive and time consuming to design. Semi-custom methodology, or standard cell based design, involves using a library containing a fixed number of standard cells such as logic gates with different fan-in and fan-out counts to implement the design. This approach yields more time savings as design automation tools such as logic synthesis tools and automating placement and routing tools can be used to generate the schematic from ta high level description language such as VHDL or Verilog, significantly reducing the design effort. However, performance of the design can lag behind custom design as fine-tuning the design is harder due to the limited number of gates in the standard cell library.

Standard cell based methodology is now the dominant approach to circuit design, with the majority of integrated circuits in production today built using this approach. For example, in the Intel Pentium ® 4 Processor, released in the year 2000, only the phase locked-loops and the clock buffers were designed manually, with virtually all other portions designed using standard cell methodology[31]. More modern processors, such as a RISC-V processor built using 45nm Silicon on Insulator technology[23], are designed and implemented entirely using the standard cell methodology. The RISC-V processor design involves writing the RTL in a high level language which is synthesized to Verilog, and using Synopsys physical design tools to map the Verilog to a 45 nm standard cell library. The processor design consists of 425K standard cells.

In the rest of this chapter, we take a look at the standard cell ASIC design flow and

Electronic Design Automation tools for the standard cell methodology.

2.1 Standard Cell ASIC Design Flow

The standard cell ASIC design flow, which is the name given to the series of steps to realize a taped out circuit ready for fabrication from a behavioral description of the circuit in a hardware description language, consists of the steps shown in figure 1 and briefly described below.

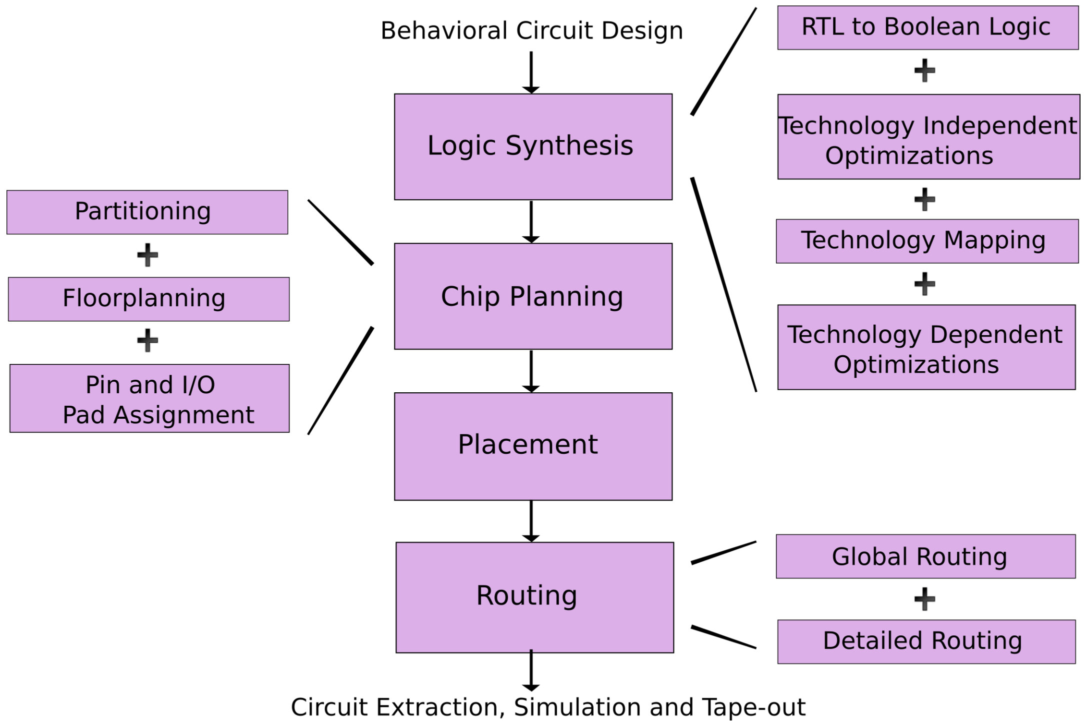

Behavioral Circuit Design
Logic Synthesis
Chip Planning
Placement
Routing
Circuit Extraction, Simulation and Tape-out
RTL to Boolean Logic
Technology Independent Optimizations
Technology Mapping
Technology Dependent Optimizations
Partitioning
Floorplanning
Pin and I/O Pad Assignment
Global Routing
Detailed Routing

Figure 2.1: Standard Cell ASIC Design Flow Steps

Logic Synthesis Once a circuit is designed, logic synthesis translates the Register-Transfer Level(RTL) behavioral description of the circuit to a structural model, initially comprising of generic gates. After this step, a series of optimization steps are performed to improve the delay, power and area metrics of the circuit. For standard cell methodology, multi-valued logic minimization[6] and the use of Boolean Satisfiability for formal verification of the synthesized circuit are the popular optimization strategies used.

The next step in logic synthesis involves a transformation to an electrical representation by mapping the generic gates to a particular implementation architecture, such as the logic libraries, embedded memories etc available for GLOBALFOUNDRIES 28-nm technology. A series of further optimization steps are performed. The resulting output is a netlist, a network of gates in the implementation technology.

Chip Planning Chip planning involves identifying and planning the arrangement of blocks and large modules in a circuit, such as caches, sensors, power converters and other intellectual property cores. These modules usually have known area and positional characteristics. Chip planning is useful as it can provide an early estimate of delay, power and area of a chip, allowing designers to identify areas of improvement[20]. Chip planning consists of:

- Partitioning - When the modules in a circuit are not specified clearly, partitioning is used to identify these modules[20]. Partitioning helps to divide the design into smaller subcircuits, each of which can operate somewhat independently from the rest. The main goal of partitioning is to perform the partitions in such a way that the connections between subcircuits is minimized, minimizing delay and improving modularity. Some popular partitioning algorithms used by EDA tools are the Kernighan-Lin algorithm and the Fiduccia-Mattheyses algorithm.
- Floorplanning - Floorplanning is the arrangement of the subcircuits on the chip. The aspect ratio and position of each module is determined during this stage. Floorplanning optimization goals include a combination of minimizing the area of the bounding box consisting of all floorplan blocks, and minimizing the total wirelength. Cluster growth, where floorplan is constructed by iteratively adding blocks until all blocks have been assigned, and the simulated annealing algorithm, which will be discussed later, are widely used iterative approaches to solving the floorplanning issue.
- Pin Assignment - Pin assignment is the assignment of a location to every pin that has an external connection, such as an I/O Pad. The main objectives in Pin Assignment are the reduction in electric parasitics and wirelength.

Placement Once chip planning is completed, in the placement stage of the design flow, each standard cell is assigned a location and orientation while ensuring no overlap occurs and all design constraints are met. The optimization objectives in the placement stage include estimated total wirelength, signal delay and routing congestion minimization. To perform placement, first a cost function is devised to model the placement problem. Next the two approaches that are used are a) analytic approach -

mathematical analysis is used to find optimal value of the cost function(eg. quadratic placement, force-directed placement) and b) stochastic approach - randomized moves are used to search for a configuration with a low cost(eg. simulated annealing).

Routing The routing stage inserts wires needed to connect the placed standard cells of the circuit. The routing task is divided into two stages to manage the complexity of the task.

- Global Routing - During global routing, the routing region is broken into tiles, and connections between tiles are determined by respecting the design rules and maximizing routing utility functions, such as wirelength or critical path minimization. In this way an approximate path is found between the pins of different standard cells, and between pins of standard cells and I/O pads. The global routing tiles and interconnections are modeled as a graph, and graph search algorithms such as maze search, A* search and line search are used for both global and detailed routing.
- Detailed Routing - In the detailed routing stage, the connections between tiles discovered in the previous stage are made concrete, by assigning specific tracks, vias and metal layers to these connections.

2.2 Motivation for Optimizing Total Wirelength

In this thesis, we explore using statistical learning techniques to speed up floorplanning and placement design parameter selections that minimize total wirelength. In this section we take a deeper look at interconnect wires and the motivation for choosing total wirelength minimization as our optimization goal.

2.2.1 On-chip Interconnects

On-chip interconnect wires in todays integrated circuits introduce a number of parasitics that affect the integrated circuits behaviour. Each parasitic causes an increase in signal propagation delay and energy dissipation and adds additional noise to an integrated circuit. These parasitics are

- Resistance - Resistance in wires is directly proportional to the length of the wire and indirectly proportional to the area of the wire. In high frequencies, an additional resistive effect, skin effect is also present, which is most common in wires carrying high frequency clock signals. Copper is the most popular material used in wires.

- Capacitance - In todays circuits with multiple metal layers, capacitance has a large effect in circuits behavior, since capacitance manifests not only between a metal wire and its semiconductor substrate dielectric layer, but also between wires at different layers. Capacitance consists of parallel plate capacitance and fringing capacitance. The most common dielectric material used in integrated circuits is Silicon dioxide.
- Inductance - Inductance is also prevalent in modern integrated circuits with high clock frequencies, causing effects such as switching noise, ringing and overshoot effects, and reflection of signals [31].

As device geometries have decreased, wires have not scaled down in an ideal fashion. In particular, while local interconnections have scaled down with device geometries, global interconnections which provide connectivity between subcircuits of the integrated circuit and with I/O have lengths which are proportional to either the die size or the complexity of the circuit[31]. Total wire length on a chip is expected to amount to around 7 km/cm by the year 2020. Other studies have shown that delay of global wires increase 50 percent per year[31], in contrast to gate delay which reduces with device geometry. As a result, interconnect delays caused by parasitics have surpassed gate delay as the major source of latency in an integrated circuit.

2.2.2 Total wirelength minimization in floorplanning and placement

Total wirelength minimization is a key optimization goal in the both the floorplanning and placement stage. A shorter total wirelength between the different blocks used for floorplanning can reduce signal propagation delays and power consumption. It also improves the routability of the integrated circuit, as it increases the amount of routing resources needed to route all standard cells and pins of the chip. Finally, since long wires between floorplan blocks can lie on the critical path of the circuit, minimizing the total wirelength between blocks is desirable from a timing closure aspect of the design process, such that the clock frequency requirements are met.

For placement, all three points mentioned in the previous paragraph applies, only the interconnects are now between standard cells instead of modules of the integrated circuit.

The figure shows the effect efficient versus inefficient placement can have on the total wirelength.

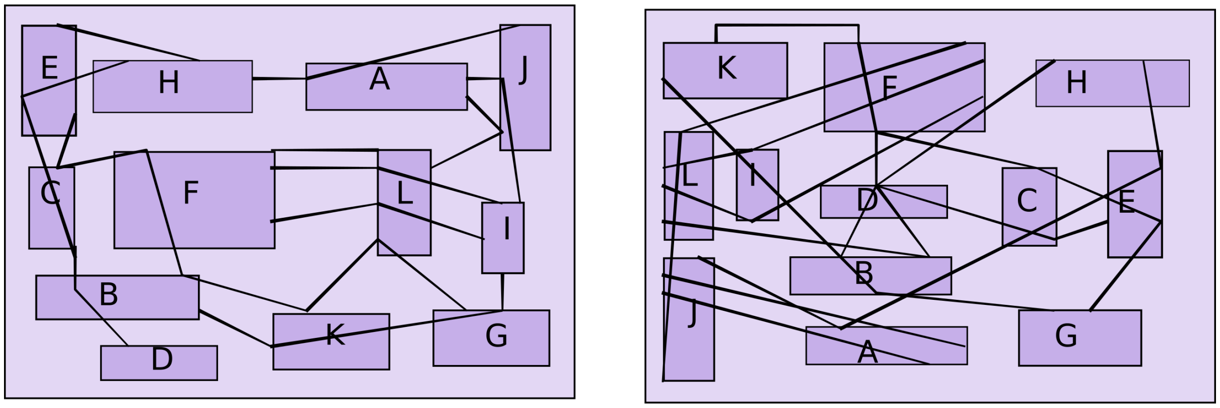


Figure 2.2: Efficient(left) vs Inefficient(right) placement of same circuit

2.3 Related Work

A number of studied have been conducted earlier that explore the use of learning algorithms to efficiently select EDA tool parameters. Wah et al [38] have explored using Genetic-based-learning to study parameter selection for Timberwolf Placement and Routing tool in order to reduce cost of placement and routing. Other studies [21] [25] have focused on using learning algorithms to efficiently select FPGA CAD tool parameters. A recent study by Ziegler et al [41] has explored using learning algorithms to tune VLSI flow parameters for the synthesis stage where the objective is to minimizing timing.

Our work is distinct from these other studies in the sense that we use statistical/machine learning algorithms to adaptively select VLSI digital flow parameters for the floorplanning and placement stages by focusing on minimizing the total wirelength.

Chapter 3

Statistical Learning Methods For Regression

In this chapter, we take a look at statistical learning methods for regression that are utilized to predict the total wirelength of an integrated circuit from its features in the next chapter.

3.1 Supervised Learning and Regression Analysis

Given a dataset containing a set of variables denoted as inputs where each input has an output as shown in equation 3.1, supervised learning is the process of learning a mapping from inputs to outputs. Based on this mapping, the goal of supervised learning is to make a prediction of the output given a certain input.

$$X = \begin{vmatrix} x_{11} & x_{12} & ... & x_{1n} \\ x_{21} & x_{22} & ... & x_{2n} \\ : & : & : & : \\ x_{m1} & x_{m2} & ... & x_{mn} \end{vmatrix} \quad Y = \begin{vmatrix} y_1 \\ y_2 \\ : \\ y_m \end{vmatrix} \tag{3.1}$$

Supervised learning is categorized into classification and regression.

- Classification - Outputs and predictions take discrete or categorical values. One example of classification is using the iris dataset [2] to predict the type of iris plant from the physical measurements of its flower, such as the petal length, height etc.

- Regression - Outputs and predictions take continuous values. One example of regression is using the abalone dataset[2] to predict the age of an abalone from its physical measurements, such as length, diameter, height and so on.

3.1.1 Regression

We focus on regression for the rest of the chapter. Regression in supervised learning can be further subcategorized as either linear or non-linear regression. A model is said to be linear when the each term in the output function is a sum of either a constant or a product of a single parameter and an input, as shown below

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... \tag{3.2}$$

On the other hand, a model is non-linear when it does not follow equation 3.2. A term can contain more than one parameter per input for example, as shown below

$$Y = \beta_0 + \beta_1 \beta_2 x_1 + ... \tag{3.3}$$

Non-linear regression is preferred for problems where a linear regression is unable to fit the data as it allows the model more degrees of freedom.

Once a regression model is trained using a training dataset, the performance of the model can be assessed by using a test dataset and evaluating the error of the prediction generated by the model compared to the actual output. The following two error measurements are typically used for regression models:

- Root Mean Squared Error - Root mean squared error measures the deviation of the actual output values from the regression prediction line. It is given by the equation 3.4

$$RMS = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - prediction(x_i))^2} \tag{3.4}$$

- Mean Absolute Error - Mean absolute error is a measure of the average of the absolute values of errors between the predictions and the actual output using a test set, given by the equation 3.5

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - prediction(x_i)| \tag{3.5}$$

3.1.2 Bias-Variance TradeOff

One of the important considerations when choosing a statistical learning model is the bias-variance tradeoff. Bias can be defined as the true error of the model used to predict the sample dataset. A model is said to have high bias if the model is too simple and does not fit the data well, leading to underfitting. Variance, on the other hand, is defined as the variability of a model for a given datapoint. High variance leads to overfitting, which means the model is too closely adapted to the training data and does not generalize well, leading to a big difference between errors on the data used to train the model, versus the data used to test the model. The expected prediction error of a regression fit for a particular input $X = x_0$ can be expressed with bias and variance terms as

$$Err(x_0) = IrreducibleError + Bias^2 + Variance[13] \tag{3.6}$$

In an ideal world, models should exhibit low bias and low variance. If a model has high bias, both training and test error will be high, while if a model has high variance, the training error will be low while test error will be high. If we try to make the hypothesis model used to predict the sample data more complicated to reduce bias, variance is increased as the model has overfit on the training data. This leads to the bias-variance tradeoff shown in Figure 3.1.

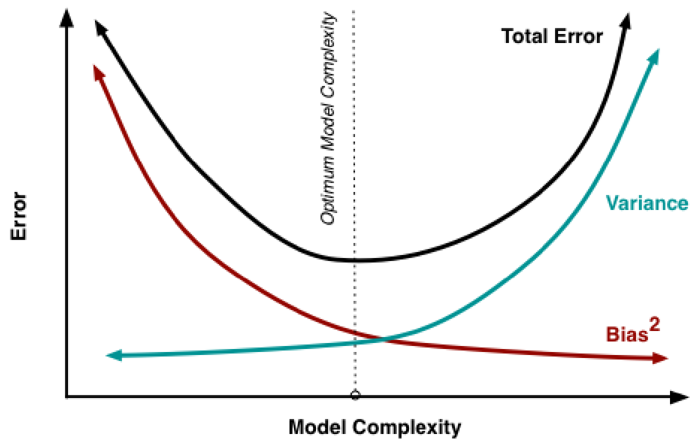


Figure 3.1: The Bias-Variance Tradeoff [12]

In the next few sections we explore some linear and non-linear models for regression that tackle the bias-variance tradeoff by using strategies such as regularization and ensembles.

3.2 Linear Regression

3.2.1 Ordinary Least Squares Regression

Ordinary least squares regression(OLS) is a common linear regression method which involves picking coefficients of the vector β that minimizes the residual sum of squares(equation 3.7) for the training set given in equation 3.1. Residual refers to the signed difference between true target y and the prediction $\hat{y}$.

$$\begin{aligned} RSS &= \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \\ &= \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{m} x_{ij}\beta_j)^2 \end{aligned} \tag{3.7}$$

Using the residual sum of squares as a cost function, a gradient descent algorithm gives a solution of predicted β as

$$\hat{\beta} = (X^T X)^{-1} X^T y \tag{3.8}$$

The full derivation of the equation is given by Hastie, Tibshirani and Friedman[13]. Once $\hat{\beta}$ is calculated, the predicted $\hat{y}$ can be calculated as $\hat{y} = X\hat{\beta}$.

3.2.2 Regularized Linear Regression

OLS regression has one limitation that reduces its usefulness in modern statistical learning. OLS regression can display high variance resulting in overfitting on the training data and capturing its idiosyncrasies, instead of generalizing. This is due to the fact that OLS does not have a mechanism to throttle back when it overfits[4]. To control overfitting, a number of methods such as subset selection (setting some input variables to 0), shrinkage (shrinking some input variables) and hybrid methods have been developed. Together these approaches are known as regularization of linear regression, as a regularization parameter is introduced to the cost function of linear regression as a control to fitting parameters.

LASSO

One approach to regularized linear regression is the Least Absolute Selection and Shrinkage Operator(Lasso)[37]. Lasso regression is formulated by ading a coefficient penalty to the OLS minimization problem in equation 3.7, such that equation 3.7 is minimized with the condition

$$\sum_{j=1}^{p} |\beta_j \leq t| \tag{3.9}$$

The bound t acts as a tuning parameter. Lasso regression balances the contradictory goals of minimizing the residual sum of squares, and minimizing the sum of absolute values of coefficients. A large value of t results in normal OLS regression. A small value of t greater than 0 results in solutions of β that are shrunken versions of what are obtained from OLS regression, or even with some coefficients set to 0. This reduces the number of input variables in the prediction, or the influence of certain input variables, resulting in lower variance. Lasso solutions can be obtained by utilizing numerical analysis or through the Least Angle Regression algorithm[13].

3.3 Non-linear Regression

When the training data consists of many features that interact with each other in complex, non-linear ways, global functions given by linear models such as OLS regression or regularized regression cannot model the data accurately. Non-linear models such as decision trees come into play for these situations.

3.3.1 Decision Trees

A binary decision tree is a representation of a function that predicts a response or a class Y from a series of inputs X_i. At each branch of a decision tree, a test is conducted on one of the inputs. Each branch leads to another branch, or to an output. Depending on the result of the test, values that resolve to true go in one direction, and false go to a different direction. Eventually at a leaf node a prediction is made which aggregates all training data points leading to that leaf node[33].

Regression Trees

A decision tree used for regression is called a regression tree. In a regression tree, the output variables of the training data are continuous valued and the prediction error is measured by taking the squared difference between the observed and predicted values. One example of a regression tree with two input variables is given in the Figure 3.2, which is models Boston housing data from 1978[15]. The goal of the tree is to predict MEDV - Median value of owner occupied homes in \$1000's. The input variables to the regression tree are:

- RM - Average number of rooms per dwelling
- NOX - Nitric oxides concentration, pphm

Using the regression tree in Figure 3.2, we can make the prediction that given the average number of rooms per dwelling is greater than 6.9 and the Nitric oxides concentration is greater than 0.66 pphm, the median value of a home was \$16,000 in 1978.

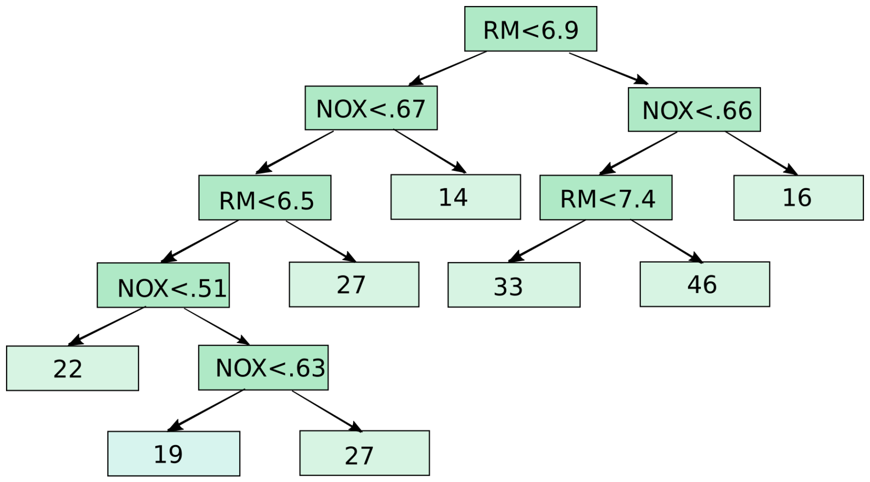


Figure 3.2: Decision Tree

Regression Tree Construction

To grow a regression tree, the algorithm needs to make a number of decisions regarding which input splitting variables j to use at branches, and the split point s for each input variable. Finding the optimal partitioning of data for a decision tree is NP-Complete[17]. As a result, popular decision tree growing algorithms use a greedy approach. For each input splitting variable, the training data is first divided into two regions $R_1(j, s) = X|X_j \leq s$ and $R_2(j, s) = X|X_j \geq s$, each region is approximated by its average value of y, and then the sum of squared error is calculated. For each splitting variable j an exhaustive search finds the split point s that minimizes this sum of squared error. This procedure is repeated for each splitting variable until j and s are found that minimize overall sum of squared error[13].

After splitting variables and points have been discovered for the regions R_1 and R_2, the algorithm continues, splitting each region in farther regions and finding the best split variables and points for each. The recursive process stops when a stopping criterion is reached, such as a given tree depth or too small a change sum of squared error. Pruning the tree is pursued in some cases to find a subtree that minimizes the sum of squared error further.

Limitation of Decision Trees

Although decision trees are a good model to capture non-linearities in the training data, they display high variance. This is due to the hierarchical nature of the tree construction process that can cause errors at the top nodes of a tree to affect the rest of the tree[27]. Decision trees display low bias, but some bias error is seen due to the greedy nature of tree construction where local cost functions are minimized. In the next few sub-sections we take a look at some techniques to reduce variance and improve the performance of decision tree based models.

3.3.2 Random Forests

Developed by Adele Cutler and Leo Breiman [7], random forests consist of a collection of decision tree models. The random forest algorithm introduces two randomizing aspects to a decision tree based model building process.

1. Random sampling with replacement - A bootstrap sample consists of a random selection of several datapoints from the training data with replacement. In random forests, a number of bootstrap samples each containing a subset of the training data are used to generate a sequence of decision tree models.

2. Random feature selection - In addition, the random forest algorithm uses a random subset of attributes for each decision tree it constructs, leaving out some features in the bootstrap sample training data.

Once a collection of M trees $f_m(x)$ are grown with these two randomizing objectives, for regression, the average of each tree is taken to obtain the resulting random forest model $f(x)$, given by

$$f(x) = \sum_{m=1}^{M} \frac{1}{M} f_m(x) \tag{3.10}$$

By averaging the trees, the variance displayed by each tree is reduced. The variance is further reduced through random feature selection in each tree, as the trees are now de-correlated from each other in a random forest. Mathematically, Equation 3.11 shows the variance of M identically distributed random variables, each with variance σ^2 and pairwise correlation ρ. Reducing pairwise correlation between the variables and increasing the number of M reduces overall variance.

$$Variance = \rho\sigma^2 + \frac{1-\rho}{M}\sigma^2 \tag{3.11}$$

3.3.3 Gradient Boosting

Gradient boosting is an ensemble tree based learning model developed by Jerome Friedman[14] to solve regression problems. Gradient boosting works by training models sequentially, starting with simple models and moving up in complexity. Each tree in gradient boosting is trained on the error left behind by all trees trained before it[4]. The set of models are combined in the end to give a complex predictor. The following example illustrates the gradient boosting algorithm.

Given a training dataset illustrated as red dots in left panel of Figure 3.3, gradient boosting algorithm initially fits a decision tree of depth 1 on the data, given by the step function in left panel of Figure 3.3. Next, the error residuals are calculated for this model, which is illustrated using green dots in the right panel of Figure 3.3. Using the error residuals as training data, another decision tree of depth 1 is used to predict the error residuals, given by the step function in the right panel of Figure 3.3.

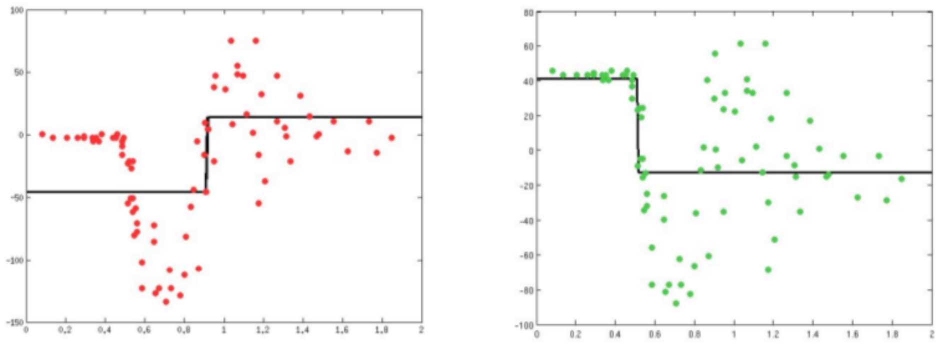


Figure 3.3: Gradient Boosting Steps 1 and 2 [18]

The two models in Figure 3.3 are added together to give a more complex model shown in the left panel of Figure 3.4. We see that the aggregated model has done a better job of predicting the training data compared to the single layer decision tree model in Figure 3.3. This series of steps is repeated, measuring the error residual after each combination, fitting a model to it and combining with earlier models until a stopping criterion is reached, such as the number of boosting stages. The overall prediction is given by a weighted sum of the collection of the individual models.

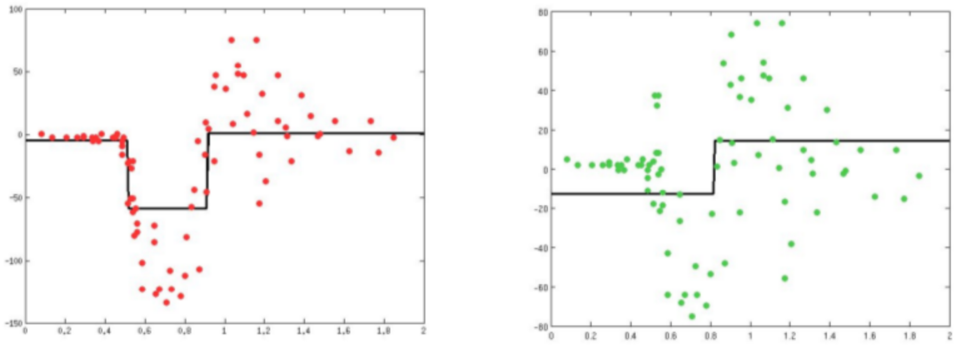


Figure 3.4: Gradient Boosting Steps 3 and 4 [18]

Chapter 4

Approach

In this chapter, we propose MachineFlow, a proof of concept web based ASIC design flow tool that allows integrated circuit designers to perform the following actions:

- Perform ASIC Design flow through an intuitive interface.
- Automate the selection of design parameters by
  - Generating training data by varying design parameter values within a range.
  - Building and using predictive models using statistical learning methods introduced in Chapter 3.

As highlighted in Section 2.2, we use total wirelength as the metric to optimize when selecting these design parameters adaptively. In particular, our statistical models are trained to minimize the **measured total wirelength after the routing step** of the ASIC flow completes, instead of an estimate of the total wirelength which is often used in existing EDA tools[20]. In the rest of this chapter, we describe the components, features and implementation details of Machineflow as well the experimental setup.

4.1 EDA Software and Design Parameters

4.1.1 Open Source ASIC Flow: Qflow

MachineFlow is based on the open source and free digital synthesis flow tool chain, Qflow, developed by Tim Edwards[9]. Using Qflow, a digital circuit defined using the high-level behavioral language Verilog can be physically laid out with a particular standard cell technology and fabrication process. Qflow does not support the cutting edge technologies supported by commercial EDA tools developed by companies such as Cadence and Synopsys, but it is a capable tool for generating the layout of smaller digital integrated circuits such as signal processing, communication, ALU circuits and so on. For example, openMSP430, a 16bit microcontroller, has been placed and routed by Qflow[9].

Qflow tool chain consists of the following components:

Design Flow Stage	Component	Function
Logic Synthesis	Yosys[40]	RTL to Boolean Logic
	ABC[5] (Included in Yosys)	Optimizations
	ABC	Technology Mapping
Chip Planning	TimberWolf[32]	Partitioning, floorplanning, pin placement
Placement	TimberWolf	Placement
Routing	TimberWolf	Global Routing
	QRouter[10]	Detailed Routing

Table 4.1: Components of Qflow, an open source ASIC Design Flow Software Package

We use a 0.35μm standard cell technology provided by the FreePDK design kit[34] in MachineFlow as it is supported by default in Qflow. The 0.35μm standard cell technology used is a 3-metal, 2-poly, n-well standard CMOS process originally developed American Microsystems Inc.

4.1.2 EDA Design Parameters Tuned Using MachineFlow

To investigate if statistical learning techniques can be used to find the design parameters that minimizes total wirelength of an integrated circuit, we focused on two design parameters of Qflow, one in the floorplanning stage and another in the placement stage.

Design Flow Stage	Parameter	Function	Default Value
Floorplanning	Number of Rows of Standard Cells	Determines the aspect ratio of the circuit	Square
Placement	Acceptance Ratio	Determines acceptance probability of new configurations	0.44

Table 4.2: Design Parameters in QFlow choosen to investigate predictive modeling

Number of Rows/Aspect Ratio

The number of rows of standard cells affects the area and shape of the global bounding box of an integrated circuit which in turn impacts the performance of an integrated circuit[20]. In Qflow, the number of rows defaults to a number that results in a square aspect ratio. Instead of using the default value, in MachineFlow we try to find an aspect ratio for the integrated circuit under design that will minimize total wirelength.

Acceptance Ratio

Acceptance ratio is a design parameter of the TimberWolf placement tool which uses simulated annealing for placement. Simulated annealing is a probabilistic method for finding the approximate global optimum of a combinatorial optimization problem. Annealing in this case refers to heating a system to a high temperature and cooling it down gradually so that ground state for the system is reached as the particles in the system rearrange in a perfectly crystallized state. In order for annealing to be effective, the cooling must be done in a way such that the particles have ample time to be rearranged and reach thermal equilibrium at each temperature involved in the cooling schedule.

In TimberWolf, to use simulated annealing for placement, a cost function C is defined whose major component is total estimated interconnect wirelength. Initially, the standard cells are randomly assigned. Next new states are generated for these cells by displacing cells, interchanging cells or changing the orientation of the cells. These new states are always accepted if change in cost $\Delta C \leq 0$, however they can also be accepted if this is not the case due to the hill climbing nature of the simulated annealing algorithm. The temperature parameter T controls the acceptance of new states. Initially during a high temperature regime, every hill climbing move is accepted. Next T is reduced in a manner that ΔC is approximately the same at each temperature during a mid temperature regime. Finally a stopping criterion is reached which gives the final placement.

The parameter acceptance ratio α comes into play during the mid temperature regime. The annealing schedule during the mid temperature regime is given by

$$s+ = s + \gamma \frac{4\alpha(1-\alpha^2)}{s^2(2-\alpha)^2\sigma^3(s)} [22] \tag{4.1}$$

Here s+ is the new inverse of temperature T. Lam et al[22] found that a value of α=0.44 was ideal to decrease the temperature while maintaining an approximate of the thermal equilibrium needed for simulated annealing. Using MachineFlow, we investigate if we can use predictive models to find a value of α for each integrated circuit under design that reduces total post routing wirelength.

4.2 Algorithm and Experimental Setup

To build predictive models to adaptively select the parameters in Section 4.1.2, we first build up a training dataset for each parameter. The training dataset consists of the following columns according to Equation 3.1:

- Input variables (X) - Consist of the design parameter plus other circuit features.
  - Design parameter - The value of the design parameter is varied within a certain range and the total wirelength is measured for these values. For number of rows of standard cells N, MachineFlow allows designers to vary N from 2 to any integer greater than 2. In our training data, we limit N to 25 rows. For acceptance ratio, MachineFlow is designed to vary the value ± 40% of the default value of 0.44.
  - Other circuit features - Other circuit features, such as the number of standard cells, global routing tracks etc. are used as other input variables in the training data. A full list of these features is given in Table 4.3.
- Dependent variable (Y) - The dependent variable is the total wirelength of the circuit measured after routing stage of the ASIC flow.

Algorithm 1 Algorithm for Design Parameter Optimization using Statistical Learning

```
 1: procedure TRAINING DATA AND FEATURE SELECTION
 2:     Choose between built-in or uploaded initial training dataset
 3:     Visually explore the chosen training dataset using boxplots and correlation heat maps
 4:     for each feature that is derived from digital flow stage earlier than stage under
        examination, and shows correlation to target of over 0.5(pearson coefficient) do
 5:         choose feature for final training dataset
 6:     Perform test-train split of training dataset and save. Randomly split data into:
            train set ← 75% of data,
            test set ← 25% of data;
 7: procedure PREDICTIVE MODEL TRAINING & DESIGN PARAMETER OPTIMIZATION
 8:     Select linear (LASSO) or non-linear (decision tree, random forests,
         gradient boosting) regression model and build model using train set
 9:     Use test set to calculate Root Mean Squared Error for chosen model
10:     Select predictive model with lowest error, M
11:     Build model M with full training data, instead of using 75% of data
12:     for each value of design parameter over its range do
13:         predict target(total post-route wirelength) using model M
14:         store design parameter and target in an associative array
15:     Find value of design parameter, V, from associative array that has smallest total
         wirelength
16:     Set design parameter to V and perform a ASIC design flow.
```

The algorithm for design parameter optimization using statistical learning is given in Algorithm 1.The training data is comprised of 30 integrated circuits with Verilog sources obtained from OpenCores [28] and other sources. A full list of ICs used is provided in Appendix A. Training data is recorded by MachineFlow into two Comma-separated values (CSV) files, one for each design parameter. The training data is also randomly split into two sets, a training set and a test set. The test set contains 25 percent of the data and is not used to train the predictive models, but is held out to evaluate the model. The test set is also used to evaluate a cross-validation score to find the best model parameters, as defined in Section 4.3.1.

The common column headers for each CSV file is given below.

	name	description
Common Features	cells	number of logic gates
	wire-bits	number of wire bits
	stdcells	number of standard cells
	global-routing-tracks	number of global routing tracks
	stdcells-after-fillers	number of standard cells with fillers
	detail-routing-tracks	number of detailed routing tracks
	total-signals	internal + input + output signals
Target	total-wire-length	post routing total wire length

Table 4.3: Common headers in Training CSV Files

Once training data is generated, correlation between features and target of training data can be examined and features which show strong correlation can be selected to build predictive models. A number of linear and non-linear regression models are available to the circuit designer. The circuit designer chooses the model that gives the **lowest error rate** for the given training data. Next MachineFlow makes a **sequence of predictions** against this model, using the design parameter varied over the allowed range as input, along with other circuit features chosen earlier that show a strong correlation to the target. The predicted values of the total wirelength and the corresponding design parameters are stored in a key value data structure. We do a search on this data structure to find the value of the design parameter that corresponds to the **lowest total wirelength**. MachineFlow is then configured to use the chosen value of design parameter, and a ASIC design flow is performed.

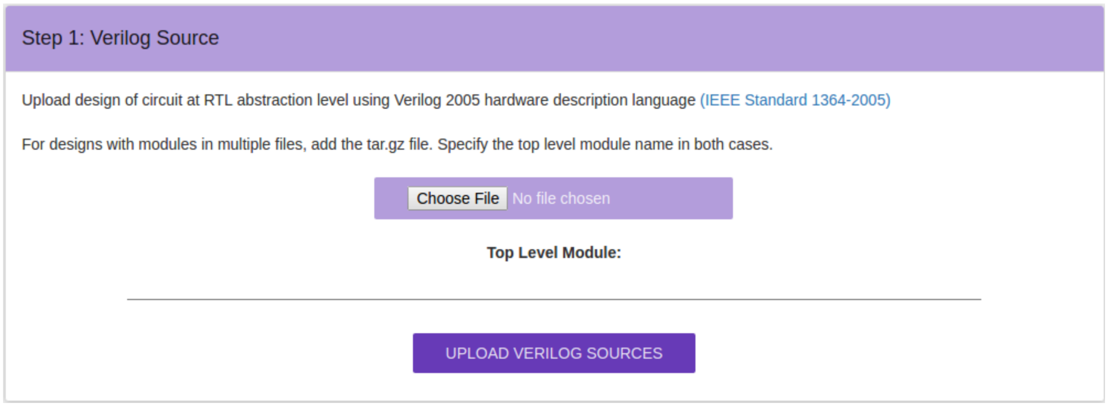
4.3 MachineFlow Web Tool

MachineFlow is used to conduct the experiment described in the previous section. We explore the features, architecture and implementation details of MachineFlow in this section.

4.3.1 Features

Simple ASIC Flow Interface

The ASIC flow interface of MachineFlow consists of a simple upload box for the design source. Users are able to submit the Register-Transfer level abstraction of their design using Verilog 2005(IEEE Standard 1364-2005) Hardware Description Language. For designs broken down into multiple Verilog modules in different files, users can upload the archived tarball compressed with gzip (tar.gz file). The top level module identifying the top module in the hierarchy needs to be specified. A screenshot of the ASIC Flow interface is shown in Figure 4.1.


Step 1: Verilog Source

Upload design of circuit at RTL abstraction level using Verilog 2005 hardware description language (IEEE Standard 1364-2005)

For designs with modules in multiple files, add the tar.gz file. Specify the top level module name in both cases.

Figure 4.1: Verilog File Upload

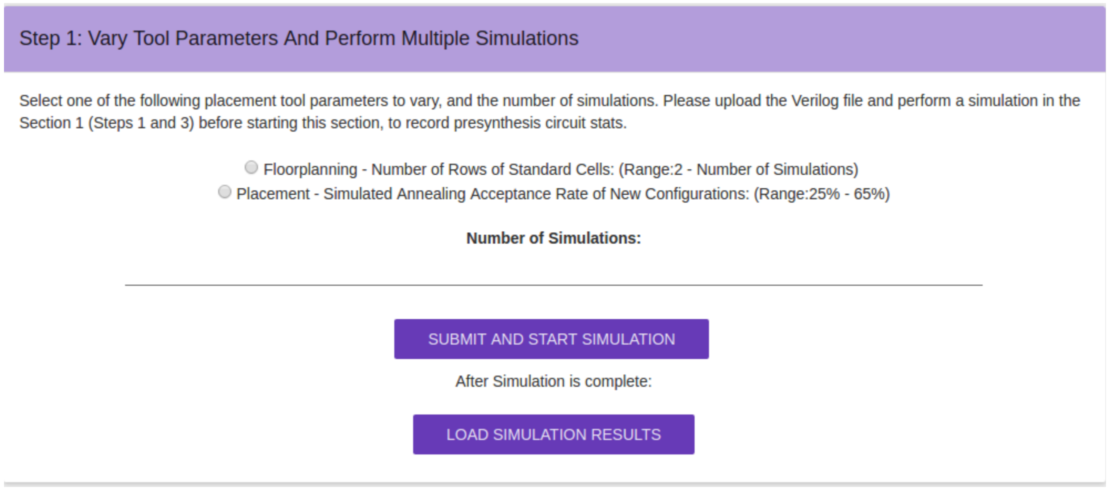
If any errors are encountered during the ASIC design flow, the error will be displayed to the user on the MachineFlow site. If no errors are encountered, the results of the ASIC design flow for the uploaded Verilog design is displayed to the user. The results obtained from ASIC design flow is tabulated in Table 4.4. Our objective is to generate a wide variety of data at different points of the design flow to aid us in building predictive models later.

Presynthesis	Synthesis	Floorplanning	Placement/Routing
num of wires	type of standard cells	rows of standard cells	standard cells with fillers
num of wires	internal signals		global routing tracks
num of wire bits	input signals		detailed routing tracks
num of public wires	output signals		routing layers
num of public wire bits	total signals		horizontal tracks
total wire bits	num of standard cells		vertical tracks
num of memories			
num of memory bits			total wirelength
num of processes			
num of cells			

Table 4.4: ASIC Flow Results

Versatile Training Data Generation

Training data can be generated by MachineFlow by choosing one of the available design parameters to tune and vary, and selecting the number of simulations to perform. The interface is shown in Figure 4.2.


Step 1: Vary Tool Parameters And Perform Multiple Simulations

Select one of the following placement tool parameters to vary, and the number of simulations. Please upload the Verilog file and perform a simulation in the Section 1 (Steps 1 and 3) before starting this section, to record presynthesis circuit stats.

○ Floorplanning - Number of Rows of Standard Cells: (Range:2 - Number of Simulations)
○ Placement - Simulated Annealing Acceptance Rate of New Configurations: (Range:25% - 65%)

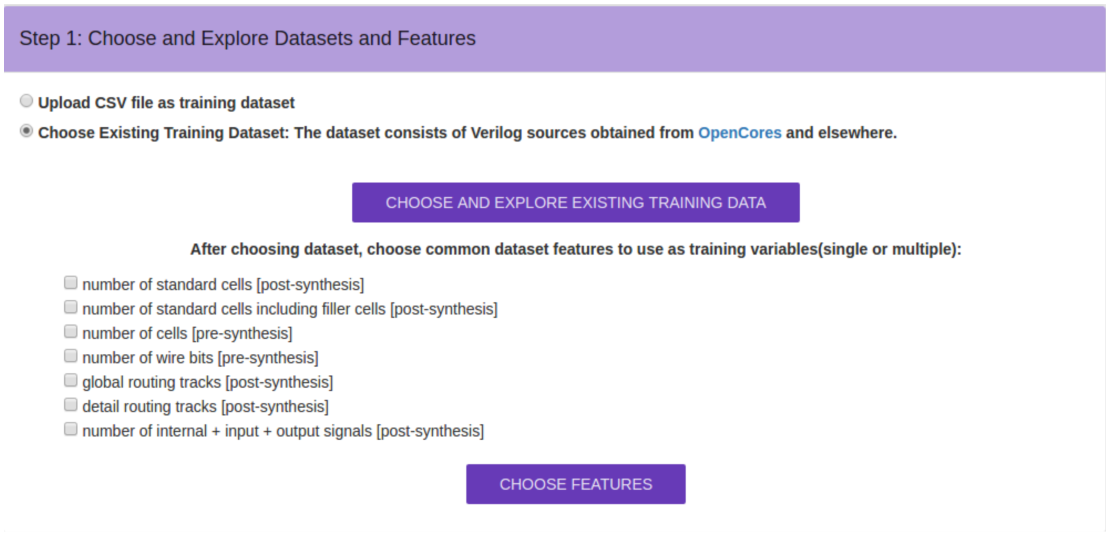
Number of Simulations:

After Simulation is complete:

Figure 4.2: Generating Training Data

The output of this section is a graph showing the value of the total wirelength for each design flow performed using a particular value of the design parameter. This graph can be used to identify the value of design parameter that results in the shortest total wirelength through a brute force search.

In addition to generating training data, MachineFlow allows users to use existing training data. Figure 4.3 shows the interface which allows users to select the circuit features they want to use alongside design parameters to generate models.


Step 1: Choose and Explore Datasets and Features

- ○ Upload CSV file as training dataset
- ● Choose Existing Training Dataset: The dataset consists of Verilog sources obtained from OpenCores and elsewhere.

After choosing dataset, choose common dataset features to use as training variables(single or multiple):

- ☐ number of standard cells [post-synthesis]
- ☐ number of standard cells including filler cells [post-synthesis]
- ☐ number of cells [pre-synthesis]
- ☐ number of wire bits [pre-synthesis]
- ☐ global routing tracks [post-synthesis]
- ☐ detail routing tracks [post-synthesis]
- ☐ number of internal + input + output signals [post-synthesis]

Figure 4.3: Choosing features

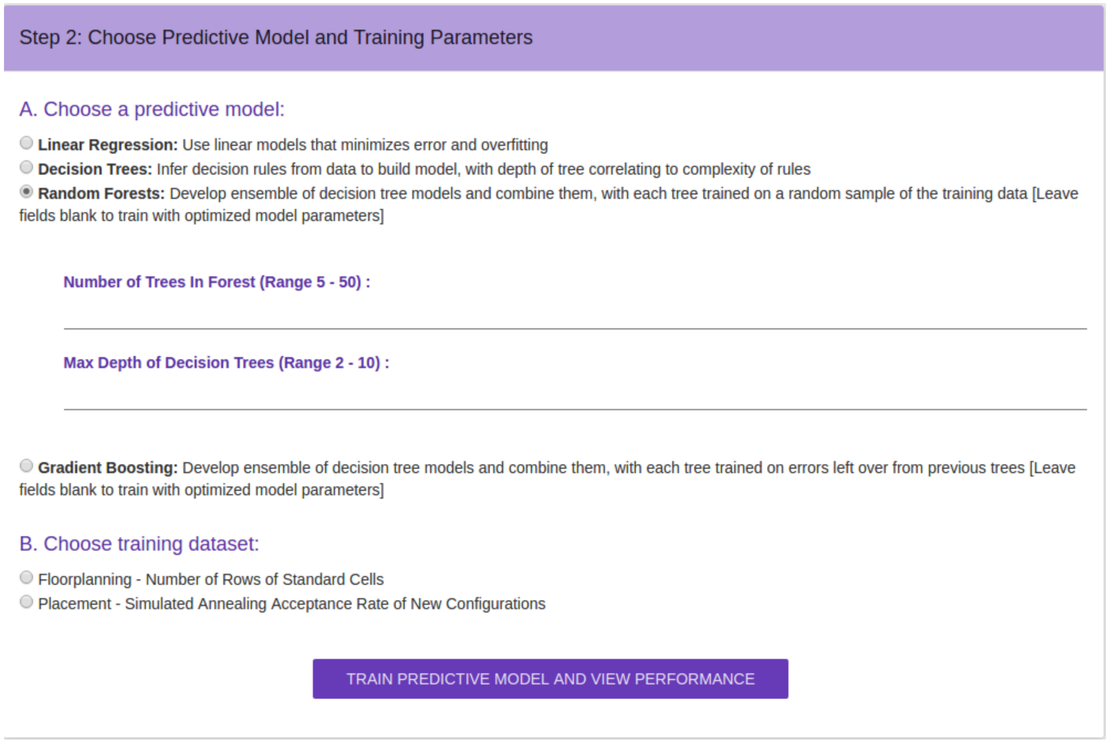
Statistical Learning Method Integration

The interface to build predictive models using statistical learning techniques is shown in Figure 4.4. Each algorithm is implemented using the scikit-learn machine learning library [30]. For certain models the user can provide parameters for the models. Table X shows the confiurable parameters for each model. If the fields are left blank for the random forests and gradient boosting models, a scikit-learn feature called Grid Search is used by MachineFlow to generate a grid of parameter values and do an exhaustive search to choose the parameters that give the best cross-validation score.

Model	Parameter	Range
Decision Trees	Maximum Depth of Decision Tree	2 - 10
Random Forests	Number of Trees In Forest	5 - 100
	Max Depth of Decision Trees	2 - 10
Gradient Boosting	Number of Boosting Stages	5 - 100
	Max Depth of Decision Trees	2 - 10
	Learning Rate of Decision Trees	0.01 - 1.0

Table 4.5: Predictive Model Parameters Available for Tuning

For the decision tree model, scikit-learn does not provide a mechanism to find the depth of the tree that minimizes root mean square error(RMSE). As a result, if the maximum depth of tree field is left blank, we measure the RMSE 3 times at each depth, average them, and choose the depth that minimizes RMSE.


Step 2: Choose Predictive Model and Training Parameters

A. Choose a predictive model:

○ **Linear Regression:** Use linear models that minimizes error and overfitting
○ **Decision Trees:** Infer decision rules from data to build model, with depth of tree correlating to complexity of rules
● **Random Forests:** Develop ensemble of decision tree models and combine them, with each tree trained on a random sample of the training data [Leave fields blank to train with optimized model parameters]

Number of Trees In Forest (Range 5 - 50) :

Max Depth of Decision Trees (Range 2 - 10) :

○ **Gradient Boosting:** Develop ensemble of decision tree models and combine them, with each tree trained on errors left over from previous trees [Leave fields blank to train with optimized model parameters]

B. Choose training dataset:

○ Floorplanning - Number of Rows of Standard Cells
○ Placement - Simulated Annealing Acceptance Rate of New Configurations

Figure 4.4: Training and Testing Models

Once a model is chosen, the root mean squared error for the model is displayed to the user for train and test data scatter plots. Root mean squared error, as given in equation 3.4 is used to assess the performance of the model.

4.3.2 Implementation

MachineFlow has been developed as a modern web application. The user interface has been implemented using HTML, Javascript, CSS and the Bootstrap front-end framework, while the backend has been implemented using Python and the Django web framework. SQLite has been used as the transactional database engine, while Gunicorn has been used as the HTTP web server and NGINX as the proxy server. Asynchronous requests are handled by the Celery distributed task queue. The workflow and architecture of MachineFlow is shown in Figure 4.5.

When a HTTP request comes in from a user's browser, it first hits MachineFlow's NGINX reverse proxy. A reverse proxy is used for load balancing, security, caching and as an additional layer of abstraction and anonymity. NGINX sends the HTTP request over to Gunicorn web server over UNIX sockets, since they are lighter and faster than TCP/IP sockets. We use Gunicorn over Django's built-in single threaded web server since Gunicorn is multi-threaded and can handle more than one request at a time. Gunicorn consists of a master process and worker processes. The master process monitors incoming requests and manages the worker processes, increasing or reducing their numbers based on the number of concurrent requests. Each worker process handles one request, with each worker basically carrying a copy of the web application to process the request.

Django HTTP request handler receives the request from Gunicorn and sends it to the URL dispatcher, which based on the URL, passes the request to the relevant view Python function. All business logic is contained within the views. For example, specific views start the ASIC design flow process, records generated data, build statistical learning models, make predictions and so on. Requests are processed by views and returned back to the user's browser. For requests which are asynchronous in nature, for example a request to perform multiple design flows while changing the design parameter inputs, the results are not returned immediately to the user, instead a real-time monitoring status for the task is shown. Asynchronous requests are placed in a task queue, where they are processed by Celery worker execution units. Once the task completes, the results are stored in the database, which is polled by Django. This result is then returned to the user.

In addition to data, MachineFlow also responds with relevant visualizations to the user. Visualizations are currently generated by the views using the Matplotlib plotting library[16] and saved as .png files on NGINX which acts as a server for static files, and served to the user on demand.

Python was chosen as the implementation language due to two reasons. First, there exist excellent data processing and statistical learning libraries in Python such as Numpy, Pandas and Scikit-learn which can be easily integrated with a Python web framework to process user requests and send responses to a browser. Secondly, Python is a simple to use interpreted, dynamically typed language allows which allows rapid iteration without compilation. This was instrumental in the fast development of the feature set of MachineFlow through quick develop-test cycles.

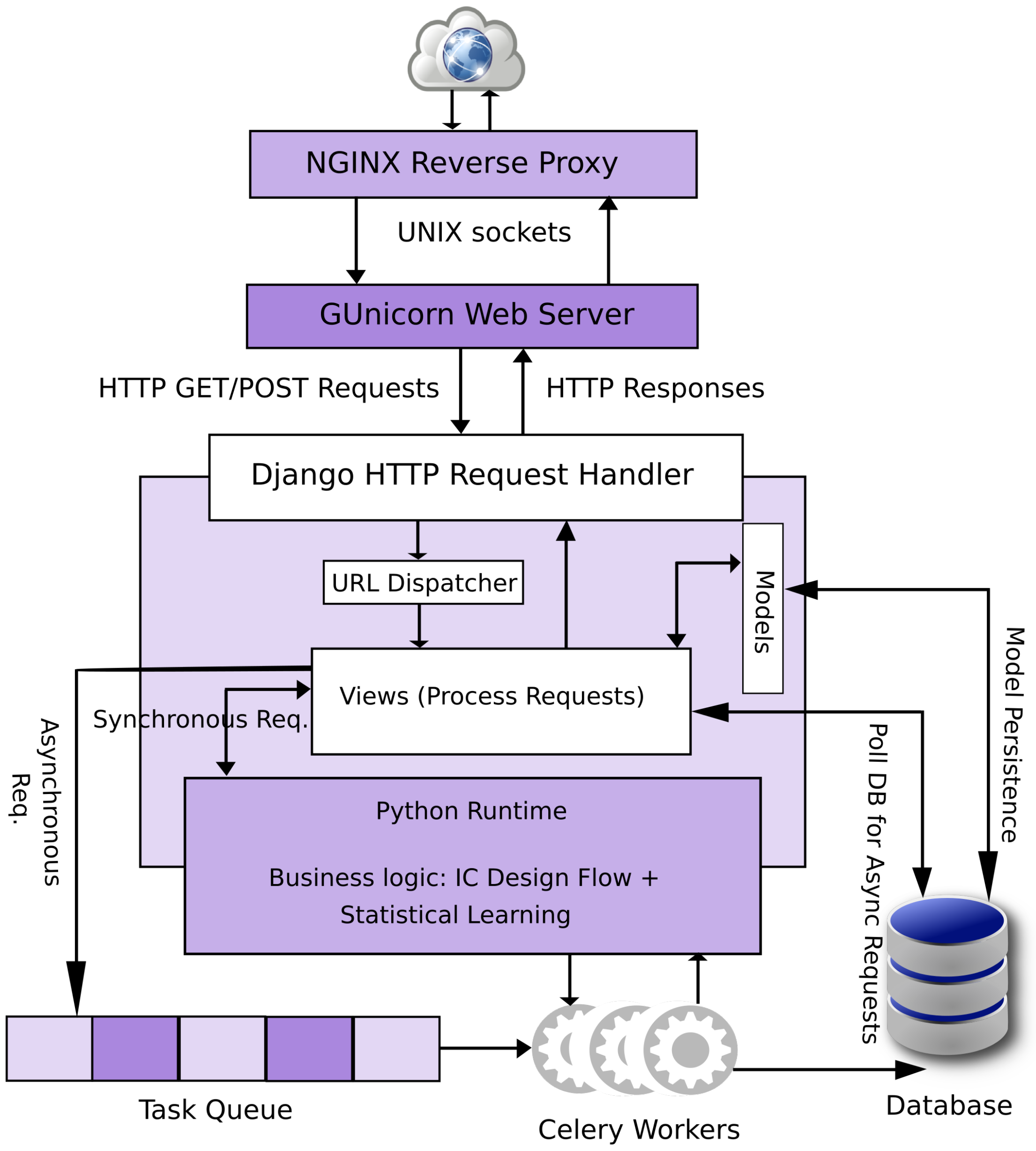


Figure 4.5: Architecture of MachineFlow

Chapter 5

Results and Analysis

We take two approaches to find the values of design parameters a) number of rows of standard cells and b) acceptance ratio described in Section 4.1.2 that minimize total wire-length. First we use predictive models to find the value of each parameter as described in Algorithm 1. Next we perform the same operation by taking a brute force approach. We use an integrated circuit which has not been included as part of the training data for these tests. We demonstrate the performance of the statistical learning approach in comparison to the brute force approach.

5.1 Statistical Learning Approach

5.1.1 Feature Selection from Training Data

The training data for each design parameter is summarized in Tables 5.1 and 5.2, showing the count, mean, standard deviation, minimum, maximum and quartile values for each column. The number of rows dataset consists of 603 records while the acceptance ratio dataset contains 502 records. This summary is shown upon choosing a training dataset in MachineFlow.

The training dataset summary page also displays a correlation matrix for each variable in the dataset to investigate the dependence between them. Each element of the symmetric matrix is a correlation coefficient between the variables i and j. We use the Pearson correlation coefficient[29] which gives the linear correlation between two variables, with a value of +1 for absolute positive correlation, 0 for no correlation and -1 for absolute

	num-of-rows	cells	wire-bits	stdcells	stdcells-with-fillers	global-route-tracks	detail-route-tracks	total-signals	total-wire-length
count	603	603	603	603	603	603	603	603	603
mean	12.48	27.47	452.75	454.8	470.68	78.53	906.66	401.43	2402922.73
std	6.17	27.59	294.65	267	269.36	39.96	545.16	236.96	1879075.01
min	2	2	81	70	71	10	135	66	210190
25%	7	6	290	237	258	46	482	205	1060535
50%	12	13	358	376	403	73	769	397	1811670
75%	17	47	634	634	650	104	1268	520	3812650
max	25	121	1310	1044	1084	198	2322	1136	9540120

Table 5.1: Summary of Number of Rows Dataset

(The acceptance ratio for this dataset is the default value of 0.44)

	acceptance-ratio	num-of-rows	cells	wire-bits	stdcells	stdcells-with-fillers	global-route-tracks	detail-route-tracks	total-signals	total-wire-length
count	502	502	502	502	502	502	502	502	502	502
mean	0.4403	9.47	28.69	355.23	382.01	393.45	62.59	769.12	344.46	2003895.07
std	0.1075	3.40	32.25	213.59	256.77	261.19	31.60	527.35	202.01	1789688.47
min	0.2667	4	2	81	64	64	17	131	66	201750
25%	0.3467	6.25	6	182	162	172	39	328	172	620220
50%	0.4400	9	13	341	335	345	54	636	349	1358305
75%	0.5333	12.75	47	413	584	601	84	1198	501	2764937
max	0.6133	17	137	1173	1044	1066	150	2184	724	7922890

Table 5.2: Summary of Acceptance Ratio Dataset

(The number of rows for this dataset is the value that gives a square aspect ratio)

negative correlation. The correlation matrix for each dataset is used for feature selection. We visually identify the correlation of the target variable (total wire length) with each circuit feature and select features that show a high correlation to the target.

Figure 5.1 shows the correlation matrix displayed as a heatmap for the number of rows dataset. The colorbar next to the heatmap maps correlation matrix value to a color, with dark red signifying a strong correlation and dark blue zero correlation. We see that the target 'total wire length' has correlation above 0.5 for the features a) standard cells b) standard cells with fillers c) global routing tracks d) detail routing tracks and e) total signals. We choose a) standard cells and b) total signals as the features for the number of rows training dataset along with the number of rows input variable. The rest of the features are not chosen since these measurements are obtained after the floorplanning stage.

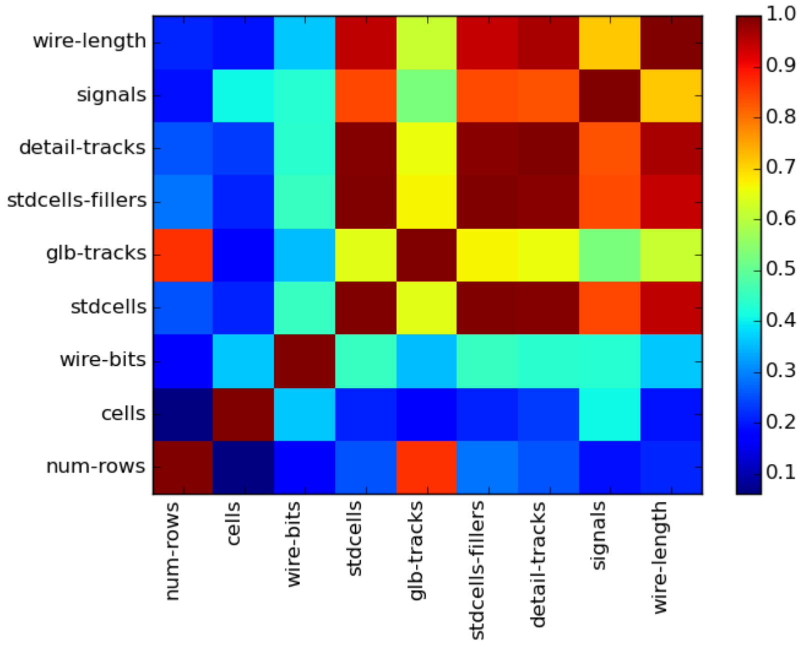


Figure 5.1: Correlation Heatmap for Number of Rows Dataset

Similarly, for the acceptance ratio heatmap in Figure 5.2, we see that the target 'total wire length' has correlation above 0.5 for the features a) number of rows b) standard cells c) standard cells with fillers d) global routing tracks e) detail routing tracks and f) total signals. We choose a) number of rows b) standard cells and c) total signals as the features for the acceptance ratio training dataset along with the acceptance ratio variable. The rest of the features are not chosen since these measurements are obtained after the placement stage.

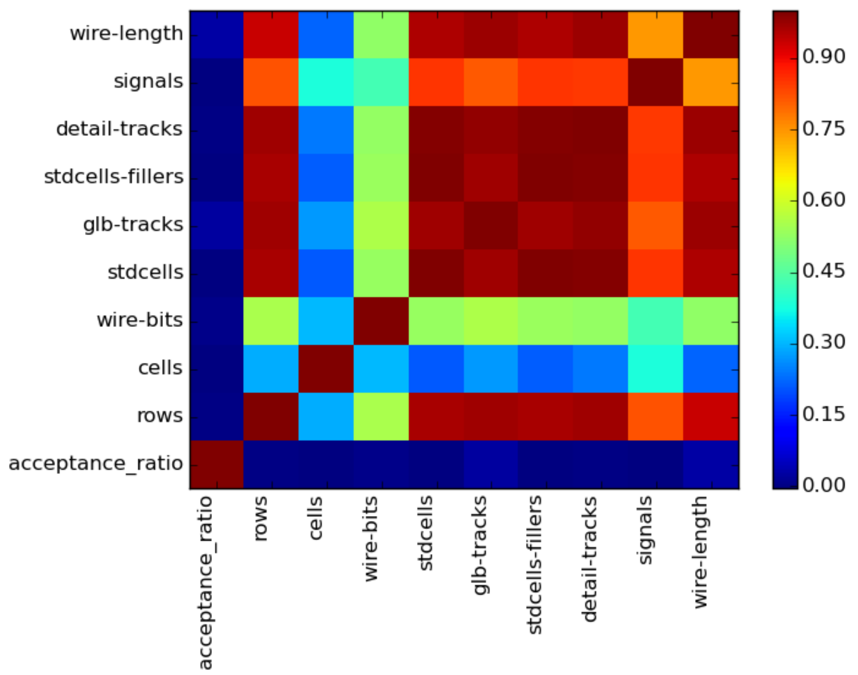


Figure 5.2: Correlation Heatmap for Acceptance Ratio Dataset

Appendix B lists boxplot and whisker plots for each feature and target variable in each dataset, which gives a visual representation of the distribution of data for these variables.

5.1.2 Model Selection

Once feature selection is done for each dataset, the final training data is generated through MachineFlow to train predictive models. As mentioned in Section 4.3.1, a train set from this training data is used to build the model while a test set is used to find the root mean squared error(RMSE) against this model. The root mean squared error shows the standard deviation between predicted and observed values for each model.

Methodology

Since the models are built from a random sample of the training data(the train split), we use the following methodology to find the RMSE.

- Perform a train-test split for each dataset, and calculate 5 RMSE values for each model using this split.
- Perform a second train-test split for each dataset, calculate 5 RMSE values for each model using this split.
- Calculate the mean of the 10 RMSE samples for each model.

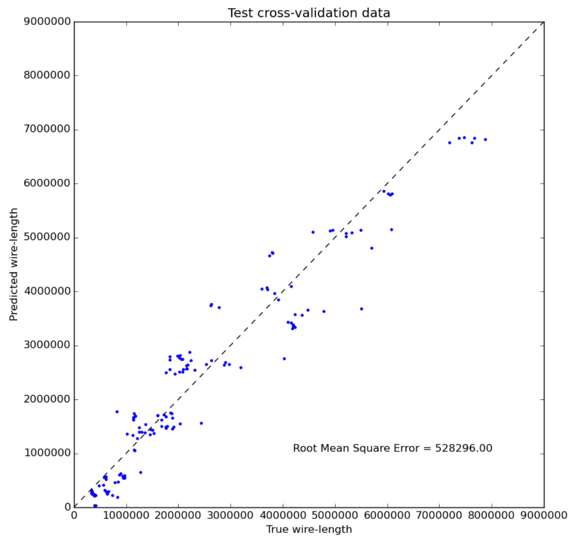
The RMSE values following the methodology above for each dataset and each model is given in Appendix B.

LASSO Regression

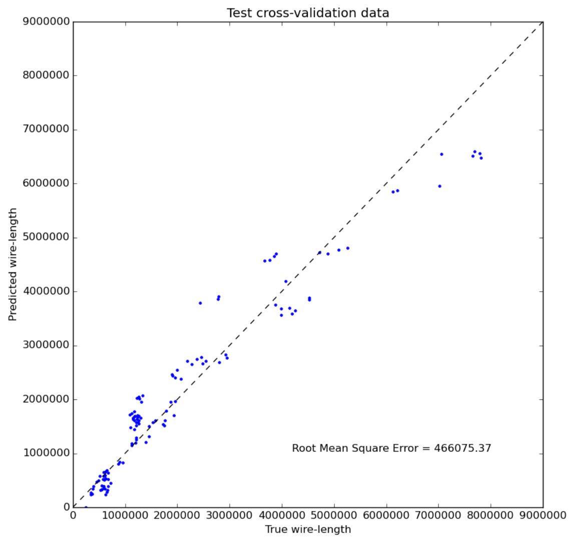
The mean RMSE values for both datasets are given in Table 5.3. Figure 5.3 and 5.4 show the scatter plot of one sample of the actual total wirelength against the predicted total wirelength given by the LASSO regression model.

Design Parameter Dataset	Samples	Mean RMSE
Number of Rows of Standard Cells	10	363516.99
Acceptance Ratio	10	253034.53

Table 5.3: LASSO: Mean RMSE Values



(a) Number of Rows Dataset



(b) Acceptance Ratio Dataset

Figure 5.3: LASSO Regression Test Scatter Plots(one sample from each dataset)

Decision Trees

The mean RMSE at each tree depth is given in Table 5.4. Scatter plots at each depth are not included for this model since to keep report concise. For the number of rows dataset, the lowest RMSE is at depth of 7 while for the Acceptance Ratio dataset the lowest RMSE is at depth 9.

Depth	Samples	Mean RMSE of Number of Rows Dataset	Mean RMSE Acceptance Ratio Dataset
2	10	621186.47	464468.78
3	10	432082.96	317023.57
4	10	292669.76	180596.03
5	10	253998.19	141299.77
6	10	236501.97	135934.92
7	10	228582.76	141498.93
8	10	235714.23	132819.57
9	10	236752.97	131743.80
10	10	240375.51	131757.68

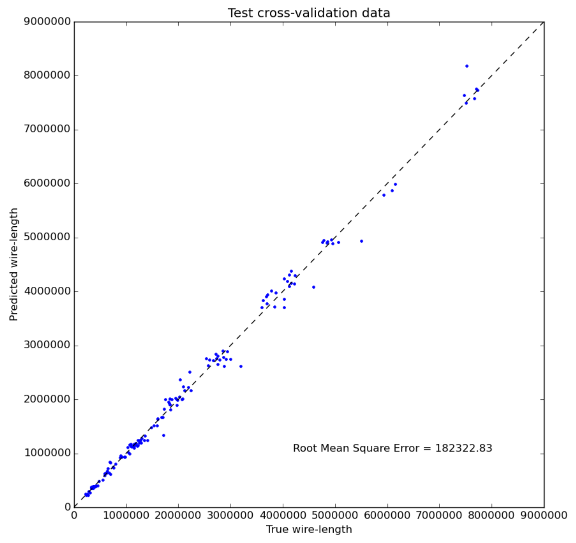
Table 5.4: Decision Trees: Mean RMSE Values At Each Maximum Depth Value

Random Forests

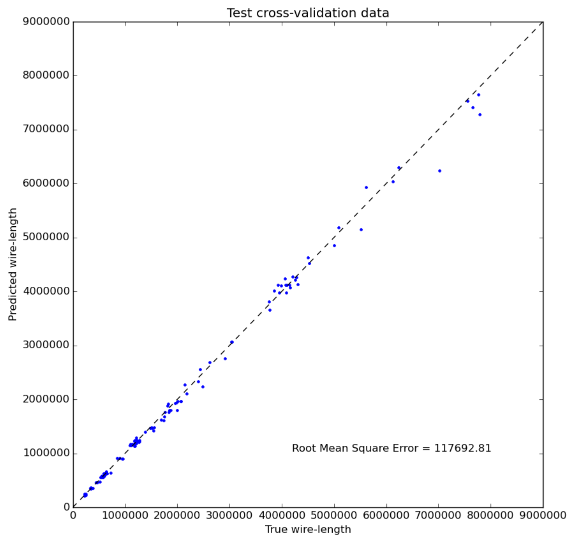
The mean RMSE values for both datasets are given in Table 5.5. Figure 5.4 shows the scatter plots of one sample of the actual total wirelength against the predicted total wirelength given by the Random Forests regression model.

Design Parameter Dataset	Samples	Mean RMSE
Number of Rows of Standard Cells	10	189066.79
Acceptance Ratio	10	117856.43

Table 5.5: Random Forests: Mean RMSE Values



(a) Number of Rows Dataset



(b) Acceptance Ratio Dataset

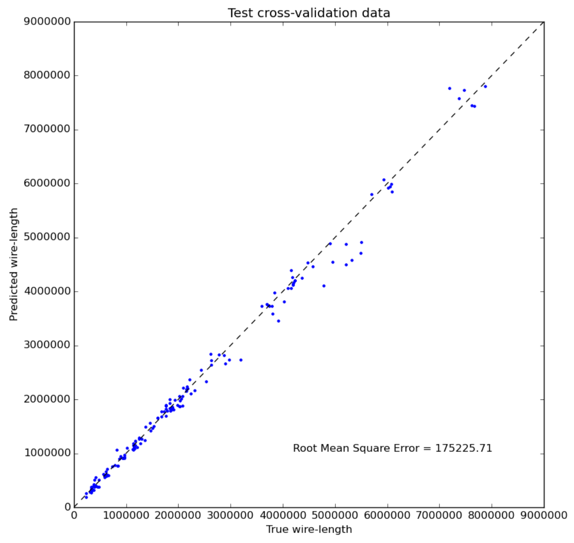
Figure 5.4: Random Forests Regression Scatter Plots(one sample from each dataset)

Gradient Boosting

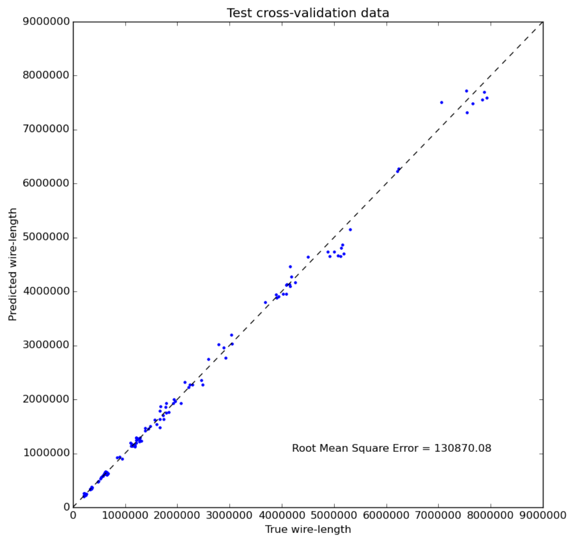
The mean RMSE values for both datasets are given in Table 5.6. Figure 5.5 shows the scatter plots of one sample of the actual total wirelength against the predicted total wirelength given by the Gradient Boosting regression model.

Design Parameter Dataset	Samples	Mean RMSE
Number of Rows of Standard Cells	10	170038.69
Acceptance Ratio	10	125797.08

Table 5.6: Gradient Boosting: Mean RMSE Values



(a) Number of Rows Dataset



(b) Acceptance Ratio Dataset

Figure 5.5: Gradient Boosting Regression Test Scatter Plots(one sample from each dataset)

Model RMSE Analysis

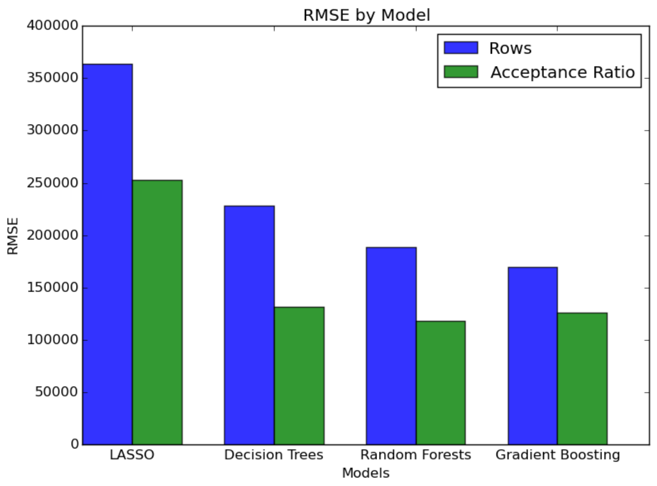


Figure 5.6: Mean RMSE values for each model by dataset

The root mean squared error for each model is compared in Figure 5.6. We make two observations:

1. Non-linear models demonstrate significantly lower RMSE compared to the linear LASSO model.
2. Among the non-linear models, Gradient Boosting model has the lowest RMSE for the number of rows dataset and Random Forests model has the lowest RMSE for the acceptance ratio dataset.

To explain the first observation, we explore the scatter plots for both datasets. To make it easier to visualize the data in 3D, we plot the total wirelength against the design parameter and one common attribute for both datasets - the number of standard cells. The hyperplane in Figures 5.7 and 5.8 represents the linear total wirelength values predicted by a linear model, while the dots represent the actual values of total wirelength for given inputs. The white dots represent observations above the plane and black dots represent observations below the plane. The color of the plane corresponds to predicted total wirelength (red - high, blue - low).

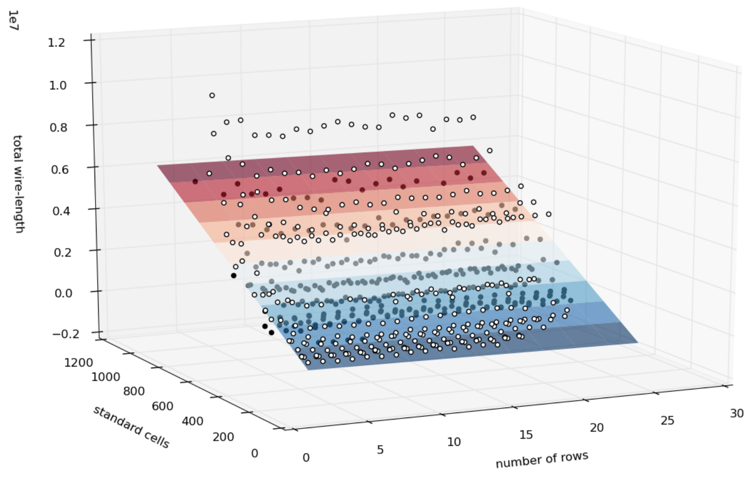


Figure 5.7: Scatter Plot for Number of Rows Dataset in Floorplanning

In Figure 5.7, we see that at lower values of number of rows, the total wirelength is higher for most sizes of circuits, with a pronounced increase for high standard cell count circuits. This can be explained through congestion caused due to limited vertical routing space. When large circuits are forced into a core with a smaller number of rows of standard cells, the efficacy of feedthrough cells used by TimberWolf placement tool in a later stage decreases as now a net can be routed through a smaller number of rows vertically. More demand is placed on horizontal routing tracks as pins that could be connected economically vertically are placed further away from each other. This can cause congestion in horizontal routing tracks and routing detours are often required. This causes total wirelength to increase as the shortest path between pins is not taken. As can be seen in Figure 5.7, a linear model given by the plane does not accurately capture the behavior of the data to account for higher wirelength at lower number of rows, which results in a higher root mean squared error for the linear model in Figure 5.6.

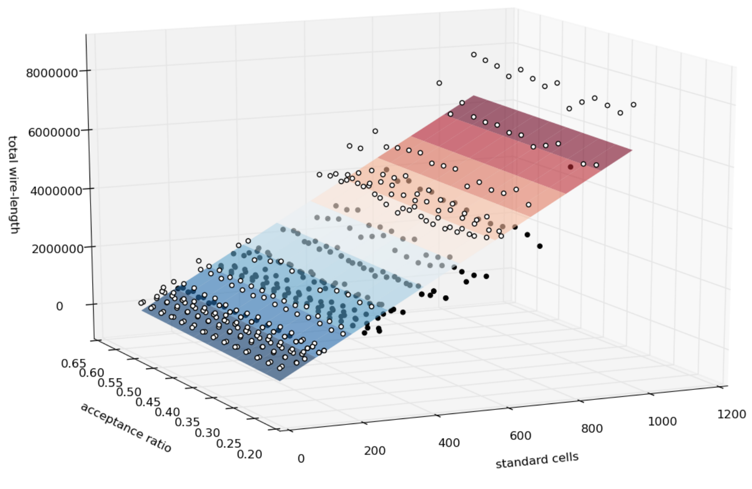


Figure 5.8: Scatter Plot for Acceptance Ratio Dataset

We also explore the acceptance ratio data distribution in Figure 5.8. We see that for circuits with higher standard cell counts, a number of minima of the total wirelength are displayed for certain inputs of acceptance ratio, while the minima are not as pronounced or not noticeable for circuits with lower standard cell counts. To explain this behavior, we revisit placement using TimberWolf tool which attempts to place cells while minimizing total estimated interconnect cost. A simulated annealing algorithm is used during placement which takes a hill climbing approach to get out of local minima and to find the global minimum estimated interconnect cost for a particular placement. Acceptance ratio is a tuning parameter for the middle stage of simulated annealing. For circuits with a smaller number of standard cells and wires, congestion is negligible as there is ample whitespace, therefore we believe the tuning parameter does not have a large effect on placement and ultimately final routed total wirelength. However, for a large circuit, there is more horizontal and vertical congestion than a smaller circuit, an effective placement given by an optimal tuning parameter can lead to less congestion and a reduction in final routed total wirelength. This is why we believe we see more pronounced minima at certain values of acceptance rates for larger circuits. Once again the linear model does not capture this behavior of the data, resulting in higher RMSE for the linear model.

Non-linear models such as decision trees described in Section 3.3 are able to better model non linear features of data by dividing the area in question into smaller sub-spaces and fitting a model in the subspace. By modeling the number of rows and acceptance ratio data features described above more accurately than a linear model, the decision tree models exhibit a lower RMSE. However, as discussed in Section 3.3.1, decision trees display high variance because the hierarchial nature of tree construction can propagate errors from the top of a tree. Random Forests and Gradient Boosting models use a collection of decision trees as described in Section 3.3.2, reducing the variance of individual trees. As a result, Random Forests and Gradient Boosting models accurately model the non-linearities in data not captured by linear models, while also reducing the variance and error in decision tree models, giving the lowest root mean squared error of the models studied.

5.1.3 Prediction Results

We choose Gradient Boosting and Random Forest models to adaptively obtain design parameters for test circuits that reduces total wirelength in this section. We use three test circuits of varying sizes. The test circuits are not part of the training data of the models.

- Small Circuit - **Twos Complement** module from the Fixed Point Arithmetic Modules project at OpenCores(http://opencores.org/project,fixed_point_arithmetic_parameterized [282 standard cells]
- Medium Circuit -**Ethernet Media Independent Interface Management** module from the Ethernet MAC 10/100 Mbps project at OpenCores(http://opencores.org/project,ethmac [640 standard cells]
- Large Circuit -**Addition** module from the Fixed Point Math Library for Verilog project at OpenCores(http://opencores.org/project,verilog_fixed_point_math_library [991 standard cells]

To compare predicted values with the actual total wirelength we first perform ASIC Design Flow runs with default design parameter values (acceptance ratio of 0.44 and square aspect ratio). For each circuit, 10 samples of total wirelength using default values are taken(Tables B.8, B.9 and B.10). The mean total-wirelength for the test circuits are given below.

- Two's Complement module - 713020
- Ethernet MIIM module - 3745690
- Addition module - 4167770

Next, we use MachineFlow to choose the random forests and gradient boosting models and find the predicted total wirelength values as described in Algorithm 1, using the button "Start Predictive Digital Flow". Since simulated annealing is a probabilistic process, the total wirelength obtained after each flow can vary. Therefore 10 samples are taken and the mean of the total wirelength is reported in the table below. If the prediction by any model chooses a default value, the result is not reported.

The predicted total wirelengths and time elapsed are summarized in Tables 5.7, 5.8 and 5.9.

Design Parameter Dataset	Model	Parameter Value Chosen By Model	Mean Total Wirelength	Samples
Number of Rows of Standard Cells	Random Forests	4	568025	10
Number of Rows of Standard Cells	Gradient Boosting	5	561330	10
Acceptance Ratio	Random Forests	0.2667	695125	10
Acceptance Ratio	Gradient Boosting	0.2667	695125	10

Table 5.7: Two's Complement Module - ASIC Design Flow Using Predictions from Statistical Models

Design Parameter Dataset	Model	Parameter Value Chosen By Model	Mean Total Wirelength	Samples
Number of Rows of Standard Cells	Random Forests	11	3218500	10
Number of Rows of Standard Cells	Gradient Boosting	default	-	10
Acceptance Ratio	Random Forests	0.3867	3405050	10
Acceptance Ratio	Gradient Boosting	0.2667	3522730	10

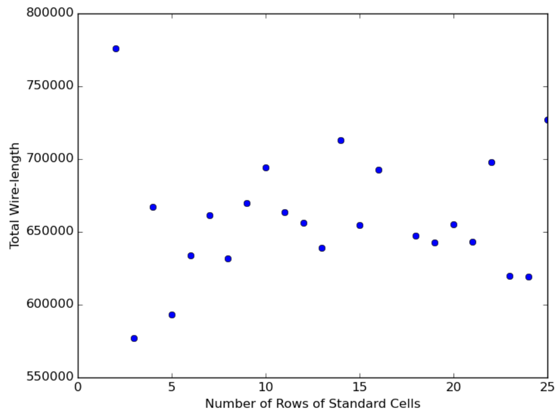
Table 5.8: Ethernet MIIM Module - ASIC Design Flow Using Predictions from Statistical Models

Design Parameter Dataset	Model	Parameter Value Chosen By Model	Mean Total Wirelength	Samples
Number of Rows of Standard Cells	Random Forests	12	3986960	10
Number of Rows of Standard Cells	Gradient Boosting	default	-	10
Acceptance Ratio	Random Forests	0.3867	4053480	10
Acceptance Ratio	Gradient Boosting	0.3867	4097835	10

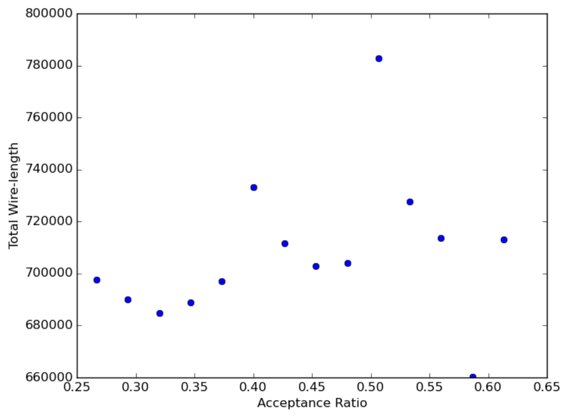
Table 5.9: Addition Module - ASIC Design Flow Using Predictions from Statistical Models

5.2 Brute Force Approach

We take a brute force approach in this section to find the design parameter values which has smallest total wirelength. Using MachineFlow, we vary the number or rows of standard cells from 2 rows to 25 rows and perform a ASIC Design Flow at each stage for all 3 circuits. Similarly for the acceptance ratio dataset, we vary the acceptance ratio from 0.267 to 0.613 and perform a ASIC Design Flow at each stage. The results are displayed in Figures 5.9, 5.10 and 5.11. The complete data is provided in Appendix B.

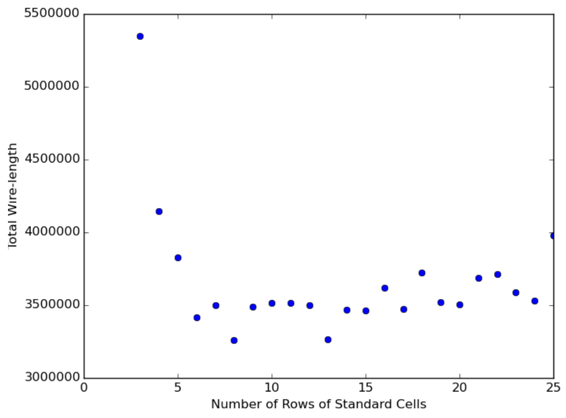


(a) Number of Rows of Standard Cells

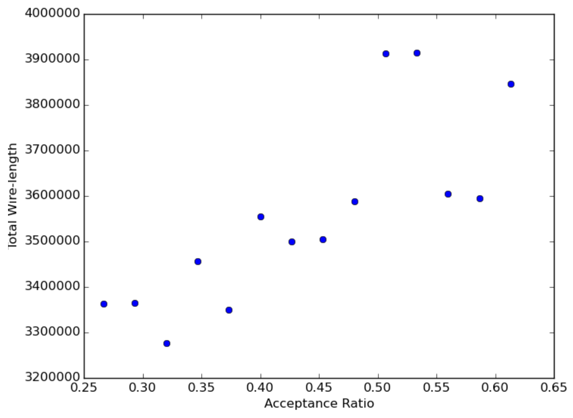


(b) Acceptance Ratio

Figure 5.9: Brute Force Exploration of Design Parameters of Two's Complement Module

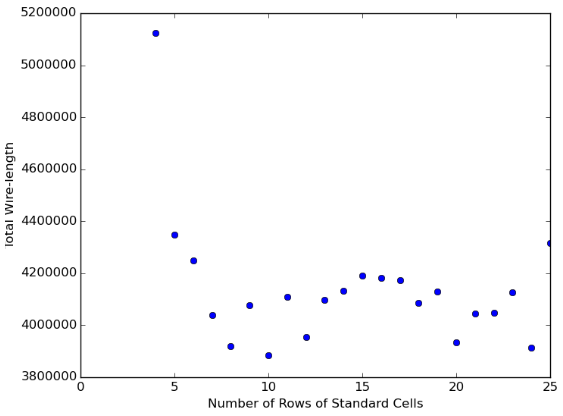


(a) Number of Rows of Standard Cells

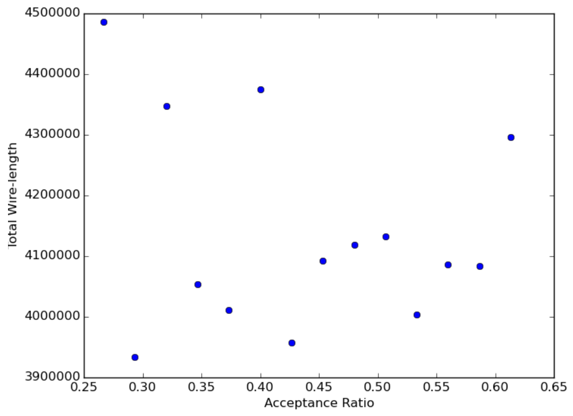


(b) Acceptance Ratio

Figure 5.10: Brute Force Exploration of Design Parameters of Ethernet MIIM Module



(a) Number of Rows of Standard Cells



(b) Acceptance Ratio

Figure 5.11: Brute Force Exploration of Design Parameters of Addition Module

Using this approach, the shortest total wirelength, the design parameter and the elapsed time for each circuit is given in Table 5.10.

Circuit	Design Parameter	Parameter Value	Total Wirelength	Time Elapsed
Two's Complement	Number of Rows of Standard Cells	3	576880	168.67s
	Acceptance Ratio	0.5867	660340	155.54s
Ethernet MIIM	Number of Rows of Standard Cells	8	3260440	561.17s
	Acceptance Ratio	0.320	3276170	350.65s
Addition	Number of Rows of Standard Cells	10	3885790	1212.74
	Acceptance Ratio	0.2933	3934040	485.24s

Table 5.10: Shortest Wirelength through Brute Force Exploration

5.3 Analysis

To verify if the total wirelength obtained using predicted design parameters is lower with statistical significance compared to the total wirelength using default parameters, we use the Mann–Whitney-Wilcoxon(MWW) test[26]. The Mann–Whitney-Wilcoxon test is used to compare two independent group of samples from continuous distributions, to check if observations in one sample are greater than the other. For each circuit, the MWW P value is calculated. If P is less than or equal to 0.05, the test indicates with a high degree of confidence that the two groups differ, thus we can say that the predicted design parameter decreased total wirelength. The P-Value for each circuit is given in Table 5.11.

Circuit	Design Parameter	Model	P-Value
Two's Complement	Number of rows	Random Forests	0.000157
		Gradient Boosting	0.000157
	Acceptane Ratio	Random Forests	0.34470
		Gradient Boosting	0.34470
Ethernet MIIM	Number of rows	Random Forests	0.00193
		Gradient Boosting	-
	Acceptane Ratio	Random Forests	0.13057
		Gradient Boosting	0.01556
Addition Module	Number of rows	Random Forests	0.00815
		Gradient Boosting	-
	Acceptane Ratio	Random Forests	0.02334
		Gradient Boosting	0.05878

Table 5.11: Mann–Whitney-Wilcoxon test results

We only consider the prediction results where the Mann–Whitney-Wilcoxon test shows the total wirelength is lower than the defaults. We obtain a P value of lower than 0.05 for the number of rows design parameter prediction for all circuits using the Random Forests model. For the acceptance ratio design parameter prediction, the Ethernet MIIM module and the Addition module circuit shows a P value of lower than 0.05 using the Gradient Boosting and Random Forests models respectively.

The percentage difference in total wire-length for the predictions passing the Mann–Whitney-Wilcoxon test and the speedup compared to a brute force search are given in Table 5.12.

Circuit	Design Parameter	Model	Difference (w.r.t. Brute Force)	Speedup(w.r.t Brute Force)
Two's Complement	Number of rows	Random Forests	0.14 %	16x
		Gradient Boosting	3.42 %	15x
Ethernet MIIM	Number of rows	Random Forests	1.21 %	25x
	Acceptane Ratio	Gradient Boosting	5.49 %	17x
Addition Module	Number of rows	Random Forests	3.70 %	35x
	Acceptane Ratio	Random Forests	3.73 %	15x

Table 5.12: Percentage Differences and Speedups

As observed in Table 5.12, the number of rows design parameter can be accurately predicted using MachineFlow framework for all three circuits with a percentage difference with respect to brute force below 5% using Random Forests Model, resulting in a speedup of 16-35x over the brute force approach. The acceptance ratio parameter could only be predicted for the large circuit under test, with a percentage difference below 5%.

The scatter plot for acceptance ratio dataset in Figure 5.8 provides a clue as to why we cannot predict the acceptance ratio for small and medium sized circuits with an acceptable percentage difference. We see that the data shows very little correlation between acceptance ratio and total wirelength for circuits with small and medium number of standard cells. The average value of total wirelength changes very little in response to changes in acceptance ratio. The probable cause for this behavior is explained in Section 5.1.2 on page 41. As a result, acceptance ratio is not a good feature for models using total wirelength as a target unless the circuit is large.

Thus our approach is capable of giving accurate and timely results for some but not all design parameters and a careful analysis of dataset, models and predictive performance should be conducted for each design parameter we want to predict.

Chapter 6

Conclusions

6.1 Summary of Contributions

This thesis describes an end to end web tool used to build and investigate statistical learners and predict design parameters of standard cell VLSI design flows. Using datasets built from 30 Verilog circuit sources, and with a target of minimizing the total-wirelength, a number of linear and non-linear models are built and evaluated to find the model that demonstrates the lowest root mean squared error values. Next an empirical evaluation is made with 3 real world circuits against these models to verify if the chosen models are capable of predicting design parameters with high accuracy and time efficiencies versus a brute force approach. We demonstrate a methodology to test if our approach is able to accurately predict a particular design parameter with confidence. Using this methodology, we demonstrate that we are able to predict a design parameter from the floorplanning stage for all classes of circuits under test using our approach.

6.2 Future Work

We would like to evaluate a number of different areas in the future.

- Neural networks/deep learning - Neural networks are a powerful learning method. Recent advances in neural networks, such as deep learning and the release of open source tools such as TensorFlow provides us with one other powerful learning technique that can be used to predict circuit design parameters.

- Larger/More diverse datasets - For the design parameters studied in this thesis, our dataset consisted of approximately 500 datapoints from 30 circuits obtained mostly from OpenCores. In the future, we would like to build a dataset with 100+ circuits and 1000+ datapoints from more diverse sources to create more rounded datasets.
- More features/targets - We would like to evaluate if we can extract more finer grained information from a standard cell VLSI design flow to test whether they would be good features showing strong correlation to targets. We would also like to explore new targets apart from minimizing total wirelength, such as minimizing the maximum interconnect delay, or minimizing delay in critical nets.
- New design parameters - We would also like to evaluate more design parameters from other stages of the flow, such as the Routing stage.

References

[1] Semiconductor Industry Association. *National technology roadmap for semiconductors*. SIA, 1997.

[2] Catherine Blake and Christopher J Merz. {UCI} repository of machine learning databases. 1998.

[3] Bill Bowhill, Blaine Stackhouse, Nevine Nassif, Zibing Yang, Arvind Raghavan, Charles Morganti, Chris Houghton, Dan Krueger, Olivier Franza, Jayen Desai, et al. 4.5 the Xeon® processor E5-2600 v3: A 22nm 18-core product family. In *Solid-State Circuits Conference-(ISSCC), 2015 IEEE International*, pages 1–3. IEEE, 2015.

[4] Michael Bowles. *Machine Learning in Python: Essential Techniques for Predictive Analysis*. John Wiley & Sons, 2015.

[5] Robert Brayton and Alan Mishchenko. ABC: An academic industrial-strength verification tool. In *Computer Aided Verification*, pages 24–40. Springer, 2010.

[6] Robert K Brayton, Gary D Hachtel, Curt McMullen, and Alberto Sangiovanni-Vincentelli. *Logic minimization algorithms for VLSI synthesis*, volume 2. Springer Science & Business Media, 1984.

[7] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[8] Erik Brunvand. *Digital VLSI chip design with Cadence and Synopsys CAD tools*. Addison-Wesley, 2010.

[9] Tim Edwards. Qflow 1.0.90, 2016. http://opencircuitdesign.com/qflow/, accessed 09-May-2016.

[10] Tim Edwards. Qrouter 1.3, 2016. http://opencircuitdesign.com/qrouter/, accessed 20-March-2016.

[11] Eyal Fayneh, Marcelo Yuffe, Ernest Knoll, Michael Zelikson, Muhammad Abozaed, Yair Talker, Ziv Shmuely, and Saher Abu Rahme. 4.1 14nm 6th-generation Core processor SoC with low power consumption and improved performance. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 72–73. IEEE, 2016.

[12] Scott Fortmann-Roe. Understanding the bias-variance tradeoff, 2012. `http://scott.fortmann-roe.com/docs/BiasVariance.html`, accessed 30-April-2016.

[13] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[14] Jerome H Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.

[15] David Harrison and Daniel L Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of environmental economics and management*, 5(1):81–102, 1978.

[16] John D Hunter et al. Matplotlib: A 2d graphics environment. *Computing in science and engineering*, 9(3):90–95, 2007.

[17] Laurent Hyafil and Ronald L Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[18] Alexander Ihler. Ensembles of learners, Lecture Notes in Machine Learning and Data Mining, University Of California, Irvine, 2012.

[19] X Jiang, A Thomsen, P Malcovati, and MC Frank Chang. EE1: Class of 2025-Where will be the best jobs? In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 518–518. IEEE, 2016.

[20] Andrew B Kahng, Jens Lienig, Igor L Markov, and Jin Hu. *VLSI physical design: from graph partitioning to timing closure*. Springer Science & Business Media, 2011.

[21] Nachiket Kapre, Harnhua Ng, Kirvy Teo, and Jaco Naude. Intime: A machine learning approach for efficient selection of fpga cad tool parameters. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 23–26. ACM, 2015.

[22] Jimmy Lam and Jean-Marc Delosme. Performance of a new annealing schedule. In *Proceedings of the 25th ACM/IEEE Design Automation Conference*, pages 306–311. IEEE Computer Society Press, 1988.

[23] Yunsup Lee, Andrew Waterman, Rimas Avizienis, Henry Cook, Chen Sun, Vladimir Stojanovic, and Krste Asanovic. A 45nm 1.3 ghz 16.7 double-precision gflops/w risc-v processor with vector accelerators. In *European Solid State Circuits Conference (ESSCIRC), ESSCIRC 2014-40th*, pages 199–202. IEEE, 2014.

[24] Wei-Yin Loh. Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1):14–23, 2011.

[25] Azamat Mametjanov, Prasanna Balaprakash, Chekuri Choudary, Paul D Hovland, Stefan M Wild, and Gerald Sabin. Autotuning FPGA design parameters for performance and power. In *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, pages 84–91. IEEE, 2015.

[26] Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60, 1947.

[27] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[28] OpenCores. Opencores, 2016. http://opencores.org/, accessed 15-May-2016.

[29] Karl Pearson. Contributions to the mathematical theory of evolution. iii. regression, heredity, and panmixia. *Proceedings of the Royal Society of London*, 59(353-358):69–71, 1895.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[31] Jan M Rabaey, Anantha P Chandrakasan, and Borivoje Nikolic. *Digital integrated circuits*, volume 2. Prentice hall Englewood Cliffs, 2002.

[32] Carl Sechen and Alberto Sangiovanni-Vincentelli. The TimberWolf placement and routing package. *IEEE Journal of Solid-State Circuits*, 20(2):510–522, 1985.

[33] Cosma Shalizi. Classification and regression trees, Lecture Notes in Statistics 36-350: Data Mining, Carnegie Mellon University, 2009.

[34] James E Stine, Ivan Castellanos, Michael Wood, Jeff Henson, Fred Love, W Rhett Davis, Paul D Franzon, Michael Bucher, Sunil Basavarajaiah, Julie Oh, et al. Freepdk:

An open-source variation-aware design kit. In *Microelectronic Systems Education, 2007. MSE'07. IEEE International Conference on*, pages 173–174. IEEE, 2007.

[35] Sehat Sutardja. 1.2 the future of IC design innovation. In *Solid-State Circuits Conference-(ISSCC), 2015 IEEE International*, pages 1–6. IEEE, 2015.

[36] Synopsys Inc. *IC Compiler II, The Leader in Place and Route - Now With the Power of 10X*, 2014. Synopsys Datasheet.

[37] Robert Tibshirani. Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[38] Benjamin W. Wah, Arthur Ieumwananonthachai, Lon-Chan Chu, and Akiko N. Aizawa. Genetics-based learning of new heuristics: rational scheduling of experiments and generalization. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):763–785, 1995.

[39] Laung-Terng Wang, Yao-Wen Chang, and Kwang-Ting Tim Cheng. *Electronic design automation: synthesis, verification, and test*. Morgan Kaufmann, 2009.

[40] Clifford Wolf, Johann Glaser, and J Kepler. Yosys-a free Verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, 2013.

[41] Matthew M Ziegler, Hung-Yi Liu, George Gristede, Bruce Owens, Ricardo Nigaglioni, and Luca P Carloni. A scalable black-box optimization system for auto-tuning vlsi synthesis programs.

APPENDICES

Appendix A

List of Integrated Circuits Used to Build Training Data

1. adc_driver.v
2. afifo.v
3. ahb2wb.v
4. analog_bridge.v
5. at_boot_reg_writer.v
6. average_pipeline.v
7. big_vga_hexdisp4.v
8. debouncer.v
9. display_16hex.v
10. display_mux.v
11. eth_crc.v
12. eth_receivecontrol.v
13. eth_rxcounters.v

14. eth_shiftreg.v
15. eth_txcounters.v
16. fht.v
17. LIN2ALAW.v
18. MLAW2LIN.v
19. parallel_crc.v
20. radio_controller_TxTiming.v
21. rc_motor.v
22. real_time_clock.v
23. rtcdate.v
24. sensor_controller.v
25. SerialRx.v
26. SerialTx.v
27. sockit_owm.v
28. uart.v
29. wbm_picoblaze.v
30. wb_regfile.v

Appendix B

Feature and Model Selection Data

sample	RMSE Number of Rows	RMSE Acceptance Ratio
1	499304.75	413550.06
2	499304.75	413550.06
3	499304.75	413550.06
4	499304.75	413550.06
5	499304.75	413550.06
6	528295.99	466075.36
7	528295.99	466075.36
8	528295.99	466075.36
9	528295.99	466075.36
10	528295.99	466075.36
Mean	513800.37	439812.71

Table B.1: LASSO Regression RMSE Values

depth	sample 1	sample 2	sample 3	sample 4	sample 5	sample 6	sample 7	sample 8	sample 9	sample 10
2	589187.40	589187.40	589187.40	589187.40	589187.40	653185.55	653185.55	653185.55	653185.55	653185.55
3	404095.56	404095.56	404095.56	404095.56	404095.56	460070.36	460070.36	460070.36	460070.36	460070.36
4	306279.88	306279.88	306279.88	306279.88	306279.88	279059.65	279059.65	279059.65	279059.65	279059.65
5	256360.26	256360.26	256360.26	256360.26	256360.26	251636.12	251636.12	251636.12	251636.12	251636.12
6	241525.76	241525.76	241525.76	241525.77	241525.77	231478.19	231478.19	231478.19	231478.19	231478.19
7	230769.03	230769.03	230769.03	230769.03	230769.03	226396.50	226396.50	226396.50	226396.50	226396.50
8	235883.30	235883.30	235883.30	235883.30	235883.30	235545.16	231448.83	235545.16	231448.84	231448.84
9	238951.26	238951.26	238951.26	238951.26	238951.26	234554.68	238635.38	238635.38	238635.38	238635.38
10	241050.89	241050.89	241050.89	241050.89	241050.89	239700.14	239700.14	235661.16	235661.16	239700.14

Table B.2: Decision Tree Model RMSE for Number of Rows Dataset

depth	sample 1	sample 2	sample 3	sample 4	sample 5	sample 6	sample 7	sample 8	sample 9	sample 10
2	427549.94	427549.94	427549.94	427549.94	427549.94	501387.63	501387.63	501387.63	501387.63	501387.63
3	305816.47	305816.47	305816.47	305816.47	305816.47	328230.67	328230.67	328230.67	328230.67	328230.67
4	187503.85	187503.85	187503.85	187503.85	187503.85	173688.22	173688.22	173688.22	173688.22	173688.22
5	146651.09	146651.09	146651.09	146651.09	146651.09	135948.46	135948.46	135948.46	135948.46	135948.46
6	131323.84	131323.84	131323.84	131323.84	131323.84	140546.01	140546.01	140546.01	140546.01	140546.01
7	142706.25	142706.25	142706.25	142706.25	142706.25	140291.61	140291.61	140291.61	140291.61	140291.61
8	132435.44	132435.44	132435.44	132435.44	132435.44	133203.70	133203.70	133203.70	133203.70	133203.70
9	132512.37	132512.37	132512.37	132512.37	132512.37	130975.24	130975.24	130975.24	130975.24	130975.24
10	132414.78	132414.78	132414.78	132414.78	132414.78	131100.58	131100.58	131100.58	131100.58	131100.58

Table B.3: Decision Tree Model RMSE for Acceptance Ratio Dataset

sample	RMSE	Number of trees in forest	max-depth
1	182322.82	100	8
2	188864.09	100	9
3	179311.36	100	6
4	183465.25	100	6
5	180206.33	100	6
6	193816.70	100	9
7	195183.56	100	8
8	198607.97	100	10
9	191022.92	100	7
10	197866.97	100	10

Table B.4: Random Forest Model RMSE for Number of Rows Dataset

sample	RMSE	Number of trees in forest	max-depth
1	117692.80	100	9
2	116159.14	100	7
3	114554.49	100	7
4	113807.93	100	10
5	117985.25	100	6
6	119791.03	100	7
7	119743.98	100	7
8	117447.86	100	8
9	119270.76	100	7
10	122111.07	100	7

Table B.5: Random Forest Model RMSE for Acceptance Ratio Dataset

sample	RMSE	Number of trees in forest	max-depth	learning rate
1	166842.78	100	3	0.5
2	166831.81	100	3	0.5
3	165785.86	100	3	0.5
4	166677.83	100	3	0.5
5	154261.70	100	3	0.1
6	176715.67	100	3	0.5
7	176439.65	100	3	0.5
8	175225.71	100	3	0.5
9	175745.89	100	3	0.5
10	175860.07	100	3	0.5

Table B.6: Gradient Boosting Model RMSE for Number of Rows Dataset

sample	RMSE	Number of trees in forest	max-depth	learning rate
1	121773.02	100	4	0.1
2	121773.02	100	4	0.1
3	121074.00	100	4	0.1
4	121865.98	100	4	0.1
5	121167.82	100	4	0.1
6	129763.13	100	5	0.05
7	129763.13	100	5	0.05
8	129963.80	100	5	0.05
9	130870.07	100	2	1.0
10	129956.91	100	5	0.05

Table B.7: Gradient Boosting Model RMSE for Acceptance Ratio Dataset

sample	total wire-length
1	713020
2	714970
3	714970
4	713020
5	676810
6	713020
7	676810
8	676810
9	713020
10	692870

Table B.8: Measured Total Wirelength for Two's Complement Module Circuit

sample	total wire-length
1	3740960
2	3750420
3	3366090
4	3740960
5	3750420
6	3358410
7	3750420
8	3358410
9	3796540
10	3796540

Table B.9: Measured Total Wirelength for Ethernet MIIM Circuit

sample	total wire-length
1	4270300
2	4065240
3	4065240
4	4059430
5	4065240
6	4277540
7	4270300
8	4277540
9	4065240
10	4277540

Table B.10: Measured Total Wirelength for Addition Module Circuit

sample	RF rows	RF Total Wirelength	RF Elapsed Time(s)	GM rows	GM Total Wirelength	GM Elapsed Time(s)
1	4	595180	10.13s	5	649150	10.00
2	4	595180	9.92s	5	561330	10.06
3	3	569350	10.35s	5	561330	9.71
4	4	566700	10.33s	5	561300	9.93
5	4	596030	10.18s	5	650030	9.69
6	4	566700	9.97s	5	561330	9.96
7	5	561330	10.24s	5	649150	9.63
8	4	596030	9.70s	5	561330	9.38
9	5	561330	9.97s	5	561300	10.04
10	4	569350	9.84s	5	650030	9.67

Table B.11: Prediction Results with Random Forests (RF) and Gradient Descent (GM) Models for Number of Rows Dataset for Two's Complement Circuit

sample	RF Acc-Ratio	RF Total Wirelength	RF Elapsed Time	GM Acc-Ratio	GM Total Wirelength	GM Elapsed Time
1	0.2667	697380	10.79s	0.2667	692870	10.88s
2	0.2667	697380	11.16s	0.2667	692870	10.69s
3	0.2667	692870	11.23s	0.2667	697380	11.11s
4	0.2667	692870	10.82s	0.2667	692870	10.71s
5	0.2667	697380	10.84s	0.2667	697380	11.06s
6	0.2667	697380	10.19s	0.2667	697380	10.16s
7	0.2667	692870	10.11s	0.2667	692870	10.52s
8	0.2667	697380	10.10s	0.2667	692870	9.84s
9	0.2667	692870	9.99s	0.2667	697380	10.80s
10	0.2667	692870	10.20s	0.2667	697380	9.79s

Table B.12: Prediction Results with Random Forests (RF) and Gradient Descent (GM) Models for Acceptance Ratio Dataset for Two's Complement Circuit

sample	RF rows	RF Total Wirelength	RF Elapsed Time(s)	GM rows	GM Total Wirelength	GM Elapsed Time(s)
1	11	3218500	21.79s	14	-	-
2	11	3218500	22.61s	14	-	-
3	11	3520730	21.35s	14	-	-
4	11	3431280	21.51s	14	-	-
5	11	3520730	22.40s	14	-	-
6	11	3218500	21.89s	14	-	-
7	11	3218500	23.61s	14	-	-
8	11	3218500	21.79s	14	-	-
9	11	3217930	21.59s	14	-	-
10	11	3217930	25.26s	14	-	-

Table B.13: Prediction Results with Random Forests (RF) and Gradient Descent (GM) Models for Number of Rows Dataset for Ethernet MIIM Circuit

sample	RF Acc-Ratio	RF Total Wirelength	RF Elapsed Time	GM Acc-Ratio	GM Total Wirelength	GM Elapsed Time
1	0.3867	3401550	21.74s	0.2667	3523210	20.27s
2	0.3867	3405050	22.31s	0.2667	3523210	20.23s
3	0.3867	3518340	21.05s	0.2667	3522730	20.16s
4	0.3867	3401550	22.53s	0.2667	3355780	20.77s
5	0.3867	3518340	21.13s	0.2667	3355780	20.90s
6	0.3867	3405050	20.72s	0.2667	3523210	20.40s
7	0.3867	3518340	21.24s	0.2667	3522730	20.55s
8	0.3867	3405050	20.72s	0.2667	3522730	20.51s
9	0.3867	3401550	20.71s	0.2667	3355780	20.63s
10	0.3867	3518340	21.12s	0.2667	3355780	20.61s

Table B.14: Prediction Results with Random Forests (RF) and Gradient Descent (GM) Models for Acceptance Ratio Dataset for Ethernet MIIM Circuit

sample	RF rows	RF Total Wirelength	RF Elapsed Time(s)	GM rows	GM Total Wirelength	GM Elapsed Time(s)
1	11	3807480	36.09s	13	-	-
2	12	3986960	35.81s	13	-	-
3	12	3982600	35.29s	13	-	-
4	12	4231280	36.88s	13	-	-
5	12	3986960	35.37s	13	-	-
6	12	3986960	35.68s	13	-	-
7	12	4260210	36.32s	13	-	-
8	11	4083070	36.15s	13	-	-
9	12	3982600	35.42s	13	-	-
10	12	3986960	32.10s	13	-	-

Table B.15: Prediction Results with Random Forests (RF) and Gradient Descent (GM) Models for Number of Rows Dataset for Addition Circuit

sample	RF acceptance ratio	RF Total Wirelength	RF Elapsed Time(s)	GM acceptance ratio	GM Total Wirelength	GM Elapsed Time(s)
1	0.3867	4142190	32.73s	0.3867	4167600	32.48
2	0.3067	3948930	32.05s	0.3867	4053480	32.39
3	0.3867	4167600	32.43s	0.3867	4053480	32.51
4	0.3867	4142190	32.57s	0.3867	4167600	32.53
5	0.3867	4052790	32.55s	0.3867	4053480	32.53
6	0.3867	4052790	32.64s	0.3867	4167600	39.25
7	0.3867	4053480	32.57s	0.3867	4053480	36.11
8	0.3867	4053480	32.63s	0.3867	4052790	36.09
9	0.3867	4052790	32.67s	0.3867	4167600	36.00
10	0.3867	4142190	32.49s	0.3867	4142190	36.02

Table B.16: Prediction Results with Random Forests (RF) and Gradient Descent (GM) Models for Acceptance Ratio Dataset for Addition Circuit

rows-Of-stdcells	stdcells	global-routing-tracks	stdcells-after-fillers	detail-routing-tracks	total-signals	total-wire-length
2	640	38	642	1667	89	NaN
3	640	41	642	1380	89	5347210
4	640	44	644	1310	89	4145530
5	640	46	643	1310	89	3830480
6	640	48	648	1328	89	3414390
7	640	56	644	1330	89	3501280
8	640	55	652	1310	89	3260440
9	640	67	648	1312	89	3489130
10	640	73	648	1335	89	3516220
11	640	74	660	1364	89	3513950
12	640	84	648	1331	89	3500210
13	640	82	660	1338	89	3264260
14	640	88	658	1326	89	3466790
15	640	101	663	1322	89	3463100
16	640	100	668	1361	89	3618620
17	640	107	663	1344	89	3472880
18	640	122	648	1325	89	3723450
19	640	111	667	1337	89	3522630
20	640	128	669	1349	89	3503660
21	640	134	672	1316	89	3689540
22	640	134	682	1334	89	3714350
23	640	142	678	1313	89	3589410
24	640	147	660	1313	89	3532980
25	640	158	678	1333	89	3977990

Table B.17: Raw Data of Brute Force Number of Rows Exploration

acceptance-ratio	rows	stdcells	global-routing-tracks	stdcells-after-fillers	detail-routing-tracks	total-signals	total-wire-length
0.266667	14	640	92	658	1325	89	3362760
0.293333	14	640	89	658	1324	89	3364840
0.320000	14	640	90	658	1310	89	3276170
0.346667	14	640	88	658	1336	89	3455840
0.373333	14	640	94	658	1328	89	3349350
0.400000	14	640	88	658	1334	89	3554910
0.426667	14	640	100	658	1346	89	3499710
0.453333	14	640	90	672	1331	89	3504860
0.480000	14	640	89	658	1340	89	3587670
0.506667	14	640	96	658	1321	89	3913070
0.533333	14	640	102	658	1333	89	3914590
0.560000	14	640	96	658	1321	89	3605040
0.586667	14	640	87	658	1327	89	3595380
0.613333	14	640	98	658	1324	89	3846970

Table B.18: Raw Data of Brute Force Acceptance Ratio Exploration