Chapter 5
Synopsys: Latency Prediction for On-Chip Communication

Xingang Cao[1], Kristof Cools[2], Nhung Dang[1], Yingjun Deng[1], Tieme Goedendorp[3], Anastasiia Hraivoronska[1], Amir Parsa Sadr[1], Mikola Schlottke[1], Oliver Sheridan-Methven[4], Niels van der Wekken[5], Harshit Bansal[1]

Abstract:
Chip design is a complicated and multi-step process. After the functional description is translated in a concrete semi-conductor network, the components comprising this network need to be placed on the actual chip. The challenge not only lies in fitting a very large amount of components in a relatively small area, but also in ensuring that the electrical latency of components and links does not jeopardise the target clock frequency. In this report, techniques for latency estimation are described that will allow the placement algorithm to make well informed decisions as to where to place components, even before the final links and routes have been decided on. The methods described here are combination of classification of components based on their type and connectivity, and a data driven approach for the identification of underlying patterns. The methods will be tested on real life data and compared against the current state-of-the-art. The estimation accuracy in the root-mean-square error shows promising results.

[1]Eindhoven University of Technology, The Netherlands
[2]Delft University of Technology, The Netherlands
[3]Fontys Hogeschool, The Netherlands
[4]Oxford University, UK
[5]VOR Tech, The Netherlands

5.1 Introduction

5.1.1 Outline

The structure of this report is as follows:

- Description of background and problem statement: this summarises the problem at hand. This part is the result of a shallow literature study, our internal discussions, and most importantly the input and feedback we received from SynopSys.
- The Challenge: stripped down version of what we actually should achieve, and a short outline of the various approaches we explored.
- Methodology: provides short descriptions for each strategy implemented. For each strategy, the premise and assumptions are detailed, the implementation presented, as the success (or lack thereof) reported on.
- Conclusion: summarise the advantages and disadvantages of each method, compare to the state-of-the-art. Recommend for the future directions.

5.2 The Challenge

Synopsys is one of the main suppliers of Electronic Design Automation tools. It uses complex software to design digital *integrated circuits* (ICs), also known as *chips*. There are multiple steps involved to makes sure that these can operate at the desired clock frequency.

Currently a multiple step simulation protocol is in-place, starting from a functional description of the chip, and resulting in a concrete physical design, ready to be submitted for fabrication.

Chips within a single generation are designed and built according to a certain micro-architecture. The architecture prescribes the semi-conductors used, the manufacturing capabilities and tolerances, etc.

A physical layout implementing the target functionality needs to be designed within the restrictions imposed by the micro-architecture. The design process is subdivided in a number of steps, each lowering the level of abstraction. These are in order of execution:

- Floorplanning
- Synthesis
- Placement

- Electrical Optimisation
- Routing
- Mask preparation

The most optimal outcome to each of these steps depends on the other steps. This, however, would lead to an untractable problem. In addition, the above workflow is the part of the company and sector standard workflow and is shaped by the availability of tools, the training of experts, etc.

5.2.1 The Placement Algorithm

In this project we will focus on the placement step. In this step, the semi-conductor implementation of the logic gates, clocks, and other components are assigned placement on the board. A number of restrictions apply:

- Routability: because routes can only be created on a finite number of levels, it is possible that certain placements prohibit making the connections that are required. This is a hard constraint and should be avoided at all cost
- Electrical proximity: because routes between elements have a certain capacity, they can only respond so fast to any inputs applied to them. The longer the route, the slower the response time. This leads to de facto delays in transmission and in turn may affect the maximum clock frequency at which the device can run.

The placement algorithm can be further broken down into: *(i)* ensuring that the components do not overlap in the 2D plane, *(ii)* making sure the routing does not exhaust the number of levels in the substrate and the capacity of the via holes, *(iii)* minimizing the total length of the wires between the components, and *(iv)* meeting the targeted clock frequency. The placement step is an instance of a pretty large scale problem as the chips are generally composed of more than 10 million components and connections.

The work addressed in this report will only focus on meeting (or minimising the violation of) the targeted clock frequency. In order to achieve a specified clock frequency, the on-chip delays need to be estimated as accurate and as fast as possible by effectively utilizing the computational resources. On-chip delays could arise due to several reasons. Few of them are: *(i)* there could be a delay between a change at an input and the corresponding change at the output, *(ii)* there could be a delay due to the time taken by the signal to propagate across the wire.

The placement algorithm ensures routability, compatibility of placement with e.g., the size of components etc. In order to minimise delays, the algorithm needs more information. In fact, it would need to know the outcome of the actual routing algorithm. This in practice, is not a viable approach. Instead, a heuristic is required to predict the final delay caused by routing the elements at their respective positions. Using this information, a placement is arrived at that is compatible with the desired clock frequency.

The target clock frequency is only jeopardised if one of the routes creates a delay which is larger than the clock period. It is not necessarily a problem if the average delay is quite high, as long as it is under the clock period. In fact, it is desirable to allow this, as it provides the placement algorithm with more leeway. This will allow it to focus on the other target specifications. In terms of the heuristic, it means that in first instance it is more important to correctly guess the delay caused by connections that will likely be very close to the critical delay that is the clock period. Sinks at the end of such a connection are called *critical sinks*.

Because components are connected using routes that follow either vertical or horizontal lanes on the chip's substrate, it stands to reason that if a single distance measure should be chosen as the starting point, it will be the $l^1(R^2)$ norm $|(x, y)|_1 = |x| + |y|$. Because of it's central role in what follows, it is given the more imaginitive name *Manhattan distance*.

In pseudo-code, the placement algorithm looks like:

Algorithm 1:

```
decrease smoothness ;
adjust cost weighting factors ;
set the delay function (revisit) ;
for each conjugate gradient restart (∼ 5×): do
    for each conjugate gradient iteration (∼ 50×): do
        calculate gradients using the delay function ;
        perform one CG step ;
    endfor
endfor
```

We were given data sets from two different chip design technologies. Each data set constituted a collection of electronic networks. Each network comprises a single driver component and the sink components it is connected to.

Each component is specified as (x, y, r, f, s) signifying the position (x, y) of the component, the rise/fall time (relative to a global clock) and the slack, s, which is the excess

time a signal can take to reach a specified point in the flow path. The XML files provided by Synopsis as training and verification data also contained a field called LIBCELL for each component, which refers to its type, i.e. components with a different LIBCELL are of a different nature. It is not unreasonable that such different types are treated differently by the optimisation algorithms. This provided one of the starting points for the various classification strategies in what follows.

This available information can be used to compute the delay for every connection between a driver and a sink in a given network. It should be noted that the driver sink connection can be part of a longer chain. This implies that not in all cases the delay equals the clock period minus the difference between sink rise and driver rise.

The connection between a driver and a sink is dubbed critical if the slack is negative. In this report, we aim to arrive at an accurate heuristic to predict delays along critical paths. In practice this means that in our data-driven approaches non-critical connections will simply be filtered out.

The information on rise/fall time, and the recorded slacks should be considered training data only. This leaves the collection of (x, y) coordinates of all components at a given step of the placement algorithm as the input for our delay function generator. The specifics of the placement algorithm require us to come up with a function:

$$t_D(\Delta x, \Delta y) = t_D(|x_d - x_s|, |y_d - y_s|), \tag{5.1}$$

that gives an estimate for the delay d, given the difference along the x-axis and y-axis of the positions of the driver-sink pair under consideration. This function is required to be continuously differentiable in (Δx, Δy) i.e., (belong to differentiability class, $\mathcal{C}^1$) and needs to be strictly increasing. It is, however, allowed to use a new delay estimator in each new iteration of the outer algorithm loop. This allows e.g., to account for component density etc. (see density based approach in Section 5.6).

5.2.2 Methodology

Subsequent chapter will visit each methodoly we considered on the study group days:

- M1: A classification and regression approach: this approach is based on filtering out non-critical components and then finding reasonable linear interpolants for each components type (this info together with (x, y) coordinates is available at the outset of the placement step. See Section 5.3.

- M2: Classification and quadratic regression. A quadratic regress could be more successful based on the underlying circuit dynamics. The underlying physics of

delay and responsiveness is briefly revisited. A cost functional is introduced and the results are presented. See Section 5.4.

- M3: A statistical approach based on identifying the distribution of delays in the training data. A distribution is fitted for each value of the Manhattan distance. The resulting sequence of means is then interpolated. See Section 5.5.
- M4: It can be meaningful to include global information such as the local density of components. A strategy for this has been included in this document. Because of time constraints this approach has not been field tested. See Section 5.6.
- M5: A pure data-driven predictor based on Gradient-boosting regression. This methodology achieves the best testing performance on Data-set-1 and Data-set-2 among our trials. See Section 5.7.

5.3 M1: Classification

The patterns and trends in a Manhattan-dist vs recorded delay graph are qualitative and quantitatively different for different classes of driver-sink pairs. This provides the starting point for the classification scheme introduced here. We will considering classification according to LIBCELL, NUMSINKS, and PARITY (inclusion of invertors).

5.3.1 LIBCELL

Each driver has a LIBCELL, a field hinting towards the type of component, given in the form L*number* (e.g., $L0$ $L1$, $L2$,...). The meaning of these labels is that every driver with the same LIBCELL is the same type of component. We were hoping that different components show different behaviours. Among the critical sinks this leads to five classes of components. The trends for those five classes do not seem to be very different and indeed a classification and regression approach based on LIBCELL did not improve upon the benchmark error for delay estimation. (Figure 5.1).

5.3.2 isink vs sink: Invertor parity

Every driver has a number ($n \geq 0$) of sinks and isinks. An isink is a sink that is preceeded by an odd number of inverters. Distinction between a sink and an isink is important for deciding whether we have to compare rise with rise time and fall with fall time, or rise with fall time and fall with rise time. Now if we look at the delay time plotted against

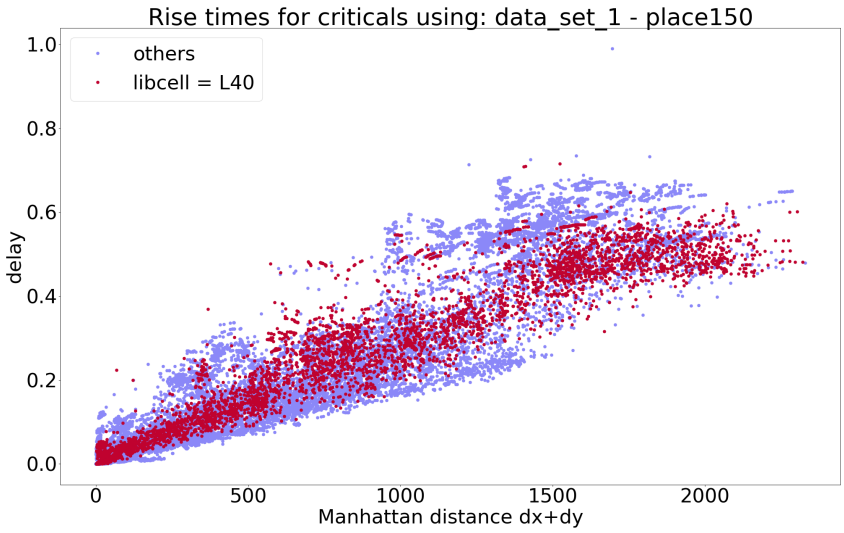

Figure 5.1: Highlighting the libcell = L40 data.

the Manhattan distance for the sinks and isinks we can see that the isinks tend to have an above average delay. See Figure 5.2.

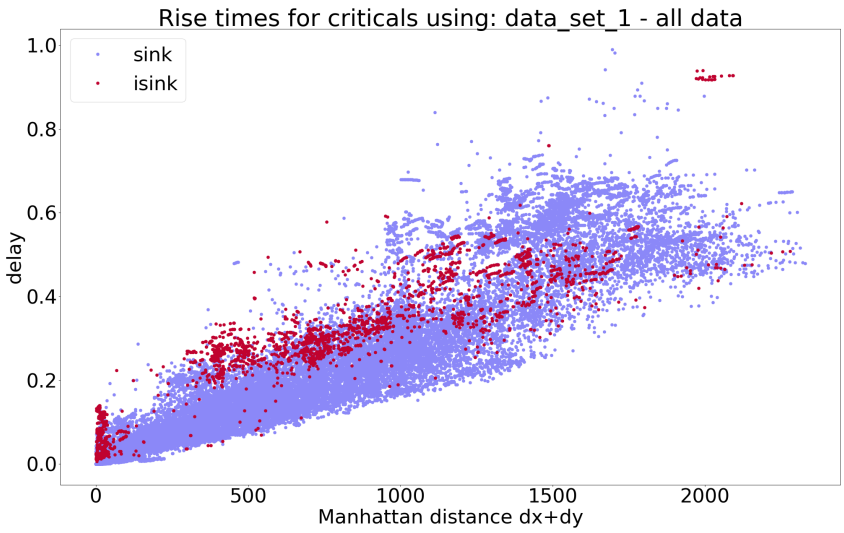

Figure 5.2: Highlighting the isink drivers.

5.3.3 Number of sinks per driver

Each component is connect to one or more other components. This network layout comes from the translation of a functional description of the chip into a set of interconnected electronic components. The graph properties of this network can be considered fixed for the problem at hand.

Some drivers have to drive only a single sink. On the other hand, some drivers have to drive ten sinks or more. Driving lots of sinks is a heavy load, and will often demand the intervention of buffers. When looking at the critical (driver, sink) pairs where the driver only has one sink, on average the delay is lower than for most (driver, sink) pairs. For (driver, sink) pairs where the driver is driving 10 or more sinks the delay is higher than on average. Improvement might be possible by further fine-tuning the exact groups, but we settled for: $n = 1$, $n = 2$, $n = 3$, $4 \leq n \leq 6$, $7 \leq n \leq 9$, $10 \leq n \leq 99$, and $100 \leq n \leq 999$.

5.3.4 Clustering of points

In the centre of the chip there is a very dense cluster of sinks. These sinks appear in the (driver, sink) pairs that have a short Manhattan distance but a relatively (very) high delay time. Filtering out this specific group of (driver, sink) pairs might result in an accurate estimate for this group, and less of an offset due to this group on the bulk data.

5.3.5 Results

Making a delay predictor function f using linear regression per category of (driver, sink) pairs has been performed. The classes of pairs used are a combination of the classification by sink vs isink and number of sinks per driver. So a total of 14 classes are defined, see Table 5.1. See Figures 5.3 and 5.4 for a selection of regression lines through 4 different categories of (driver, sink) pairs.

An interesting difference between dataset 1 and dataset 2 is that in dataset 1 both the even and odd numbered categories show a somewhat single band of datapoints. This suggests that selecting on either sink or isink neatly splits the dataset. For dataset 2 however the even numbered (sink) categories show two bands of data, where the gap in the middle is somewhat filled by the next category (isink), this is seen better for the case with a few sinks per driver. This suggests that where selecting on isink does give a single group of (driver, sink) pairs that are similar, selecting the sink (driver, sink) pairs leaves us with two distinct groups of datapoints. This could be because the lower band contains (driver, sink) pairs with 0 inverters between them, while the upper band contains (driver, sink) pairs with 2 inverters between them, resulting in an even number of inverters, and thus the 'sink' label.

Table 5.1: Classification used for the different linear regression categories.

Category 0	sink & 1 sink/driver
Category 1	isink & 1 sink/driver
Category 2	sink & 2 sinks/driver
Category 3	isink & 2 sinks/driver
Category 4	sink & 3 sinks/driver
Category 5	isink & 3 sinks/driver
Category 6	sink & 4 . . . 6 sinks/driver
Category 7	isink & 4 . . . 6 sinks/driver
Category 8	sink & 7 . . . 9 sinks/driver
Category 9	isink & 7 . . . 9 sinks/driver
Category 10	sink & 10 . . . 99 sinks/driver
Category 11	isink & 10 . . . 99 sinks/driver
Category 12	sink & 100 . . . 999 sinks/driver
Category 13	isink & 100 . . . 999 sinks/driver

5.3.5.1 RMS errors

The final result from using this method gives for the data sets, with training data the `place_150` set, and testing data the `place_100` set a root mean square error of:

Input	Error
Dataset-1	0.03881
Dataset-2	0.00844

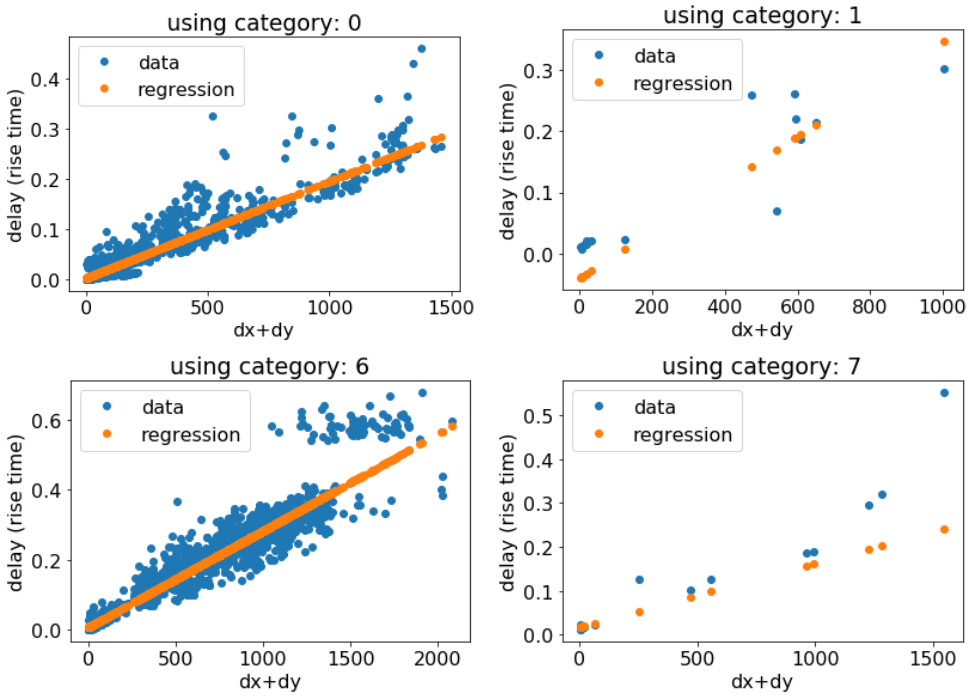

Figure 5.3: Linear regression trough subsets of dataset 1.

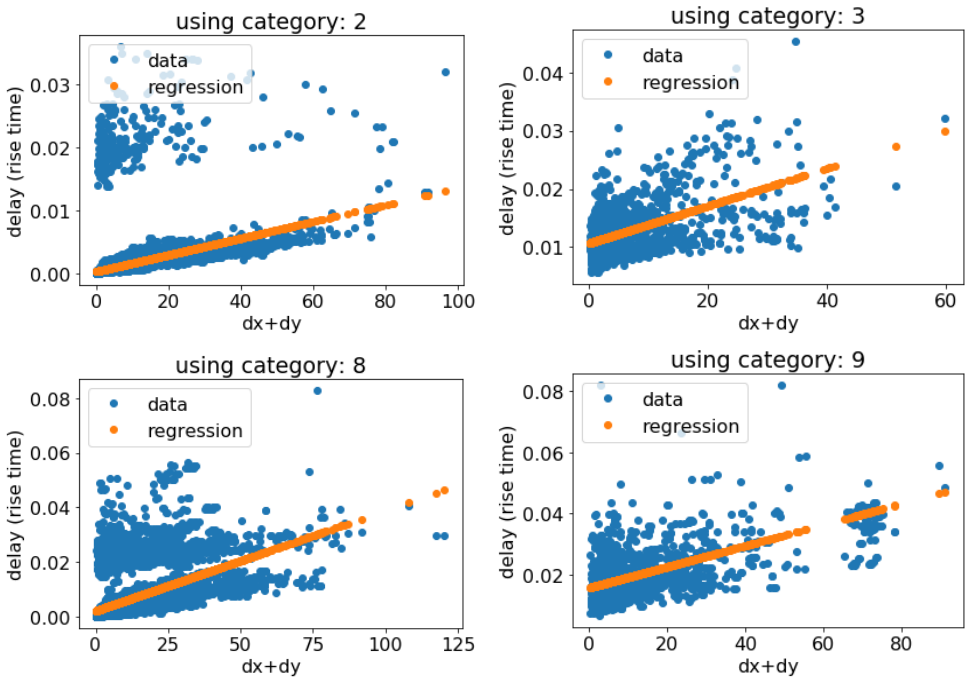

Figure 5.4: Linear regression trough subsets of dataset 2.

5.4 M2: Regression based on underlying physics, classification, and quadratic fitting

5.4.1 Physical insights of the delay between source and sinks

The latency of a signal is determined by the resistances and capacitances of the connections (or wires) and components (or cells). Both delay in cell and delay in wire usually depend on placement, routing topology, etc. In this section, we are interested in the delay in wire, which also depends on metal layer choices, neighbouring, etc. In fact, the approximate delay in wire $t_{D,wire}$ is

$$t_{D,wire} \approx R_w \times \left(\frac{C_w}{2} + C_{in}\right) = \frac{R_u C_u}{2} \times L_w^2 + R_u C_{in} \times L_w, \tag{5.2}$$

with $R_w \approx L_w \times R_u, C_w = L_w \times C_u$, where L_w [meter], R_u [Ohm/meter] and C_u [Farad/meter] represent the length of the connection from diver to sink/isink, conductance and capacitance belonging to the connection, respectively, see Figure 5.5. Moreover, when the connections are long (see Figure 5.5) or connections are from one to many points (see Figure 5.6), repeaters (or buffers) are placed to boost the signal. In this case, $t_{D,wire}$ is approximated by

$$t_{D,wire} \approx t_{D,buf} + 2 \times \frac{R_w}{2} \times \left(\frac{C_w}{4} + C_{in}\right) = t_{D,buf} + \frac{R_u C_u}{4} \times L_w^2 + R_u C_{in} \times L_w, \tag{5.3}$$

where $t_{D,buf}$ is the delay of a repeater.

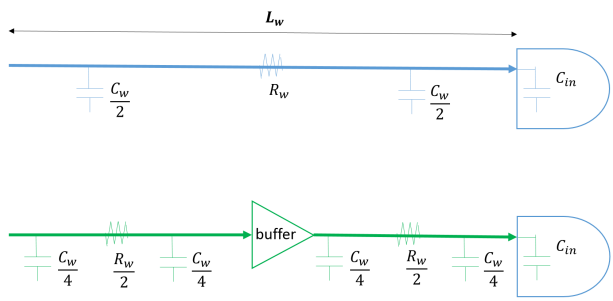

Figure 5.5: Connection from driver to sink (top) and long connection buffered by repeaters (bottom). For each case, the delay $t_{D,wire}$ is determined by (5.2) and (5.3) respectively.

From (5.2) and (5.3) of $t_{D,wire}$, we see that physically, the delay function should have quadratic behaviour, and is of the form

$$f(x_d, y_d, x_s, y_s) = \alpha_2 d_M^2 + \alpha_1 d_M + \alpha_0, \tag{5.4}$$

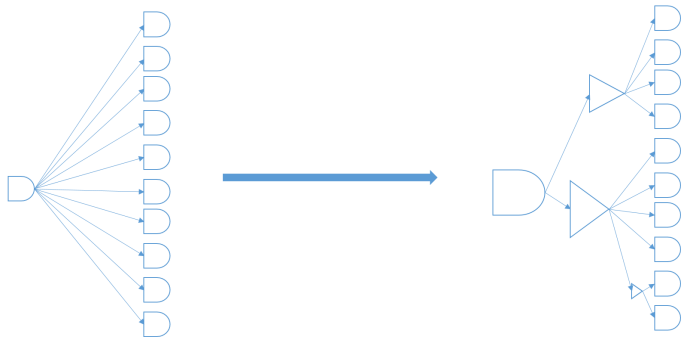

Figure 5.6: Connections are not one to one but one driver to many sinks/isinks. In this case components/cells may be sized and/or buffer may be needed.

where d_M represents the Manhattan distance between two points, with $d_M = |x_d - x_s| + |y_d - y_s|$. When the distance is long and/or the connection is not point to point, a repeater with delay α_0 is needed. $\alpha_i, i = \{1, 2\}$ are coefficients corresponding to resistances and/or capacitances from (5.2) and (5.3). We need to determine $\alpha_i, i = \{0, 1, 2\}$ to have the general formula of delay function. To aim this, a regression model is built.

Note: we realise that the electrical optimisation has as a consequency that the piecewise quadratic form described above globally resembles a linear function. This implies the approach pursued here cannot in general be successful and requires some information about the circumstances under which electrical optimisation takes place.

5.4.2 Regression model

For simplicity let $p = (\alpha_2, \alpha_1, \alpha_0)$. The piecewise function J minimizes the differences between our predicted delay function $f_p(x_d, y_d, x_s, y_s)$ and delay computed after placement (or reference delay) t_D from the data. Denote N as the number of delays computed for each pair of driver and sink/isink. A cost function can be defined as

$$J(p) = \frac{1}{N} \sum_{i=1}^{N} \left(f_p(x_d^i, y_d^i, x_s^i, y_s^i) - t_D^i \right)^2 \tag{5.5}$$

To minimize the $J(p)$, we use the data extracted from file `place_150` of data set 1. The parameters $p = (\alpha_2, \alpha_1, \alpha_0)$ of $f_p(x_d, y_d, x_s, y_s)$ are found for different types of extracted data. Precisely, we find p for

- 1 driver, multiple sinks/isinks case
- 1 driver, critical sinks/isinks case

Case	$p = (\alpha_2, \alpha_1, \alpha_0)$	Error E
1	$p = (10^{-7}, 4.8 \times 10^{-4}, 3.5 \times 10^{-3})$	0.18
2	$p = (-2.87 \times 10^{-8}, 3.54 \times 10^{-4}, -5.21 \times 10^{-4})$	0.19
3	$p = (-3.26 \times 10^{-8}, 3.81 \times 10^{-4}, 2 \times 10^{-2})$	0.36

Table 5.2: Computed values of p and error E

- 1 driver, critical sinks case

The value of computed p and E for each case are in Table 5.2. The total relative error for each case is computed by

$$E = \sum_{i=1}^{N} \frac{|t_D^i - f_p(x_d^i, y_d^i, x_s^i, y_s^i)|}{t_D^i}$$

.

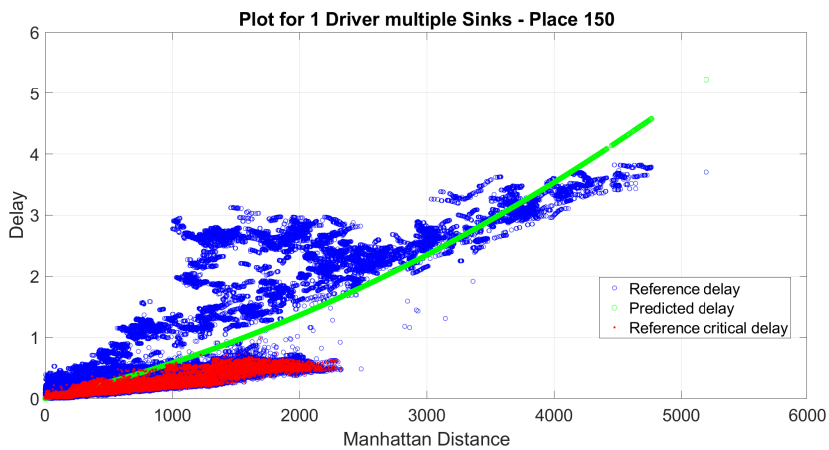

Figure 5.7: Estimated delay for case 1: 1 driver, multiple sinks/isinks

For the first case, our delay function follows the dominated shape of the reference delay t_D in Figure 5.7 which is close to what we expected. The second and third cases give results not as good as the first one since we deal with critical points (which is more challenging). In fact, latency of signal also depends on topology of the circuit, physical effects, etc., thus looking at only coordinates of driver and sink/isink would not give enough information. Extra efforts would need in the future to overcome this challenge.

5.5 M3: A Statistical Approach on Data Set 2

In data set 1, the relation between the Manhattan distance and the delay shows a clear trend. However, the relation of these two physical quantities is not very clear for the data set 2. This can be seen from Figure 5.8. Whatever the cause, the qualitative difference suggests additional data is needed in order to develop an algorithm that is efficient on multiple data sets, corresponding to different chips design and fabrication technologies. Parameters that we expect are particularly relevant are number of layers in the substrate, etching resolution, and choice of semi-conductor materials.

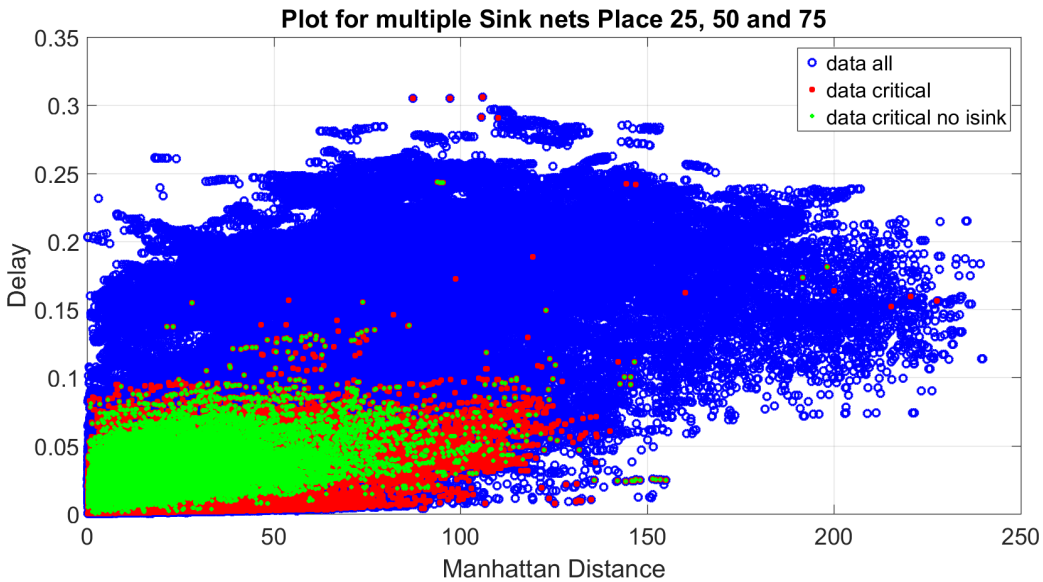

Figure 5.8: Scatter plot of `place_25`*,* `place_50` *and* `place_75` *of data set 2.*

To investigate the relation between the Manhattan distance and the delay, we attempt to first find the distribution of the data and then use the statistics behind it to model the relation between the Manhattan distance and the delay.

5.5.1 Distribution Fitting

To perform statistic analysis of the relation between the Manhattan distance and the delay, we first divide the data into small subsets. Denote the Manhattan distance as d_M and the delay as t_D. When Manhattan distance is less than 40, we divide the whole data set into 8 subsets. Namely, in subset $\mathcal{S}_i, i = 1, 2, \ldots, 8$, the Manhattan distance satisfies $d_M \in [(i-1) \cdot 5, i \cdot 5)$ and the delay t_D are the corresponding delay data at those

Manhattan distances. Then $\mathcal{S}_i$ can be written as

$$S_i := \{(d_M, t_D) | d_M \in [(i-1) \cdot 5, i \cdot 5)\}, \text{ if } d_M < 40, \ i = 1, 2, \ldots, 8.$$

When $d_M \in [40, 80)$, to make sure there is enough data in each subset, this part of data is divided into two subsets. Namely,

$$S_9 := \{(d_M, t_D) | d_M \in [40, 60)\},$$
$$S_{10} := \{(d_M, t_D) | d_M \in [60, 80)\}.$$

The last data set contains the delay and the Manhattan distance data corresponding to $d_M \in [80, 120)$, i.e.,

$$S_{11} := \{(d_M, t_D) | d_M \in [80, 120)\}.$$

Investigating the histogram of the delay t_D, one can see that the distribution to the delay in each subset can be approximated by a Beta distribution, which has the probability density function defined as

$$Beta(\alpha, \beta, x) := \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}, \tag{5.6}$$

where

$$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha + \beta)},$$

and $\Gamma(z)$ is the Gamma function. Many definitions of the Gamma function exist. For example, if $z \in \mathbb{C}^+$, i.e., a complex number with positive real part, then the Gamma function on z is defined via the following improper integral

$$\Gamma(z) = \int_0^\infty x^{z-1} e^{-x} dx.$$

If $z \in \mathbb{Z}+$, i.e., a positive integer, the Gamma function is simply the factorial of $z - 1$,

$$\Gamma(z) = (z-1)!.$$

For a more detailed explanation, we refer to the Wikipedia page of the Gamma function ([1]).

The parameters α and β in Beta distribution are used to control the shape of the function. Figure 5.9 shows how the shape of the probability density function varies according to different values of α and β. Hence, suitable choices of the parameter α and β may lead to a satisfactory fitting of the histogram of the delay data t_D in the distribution sense. Furthermore, we only consider the data points which are critical,

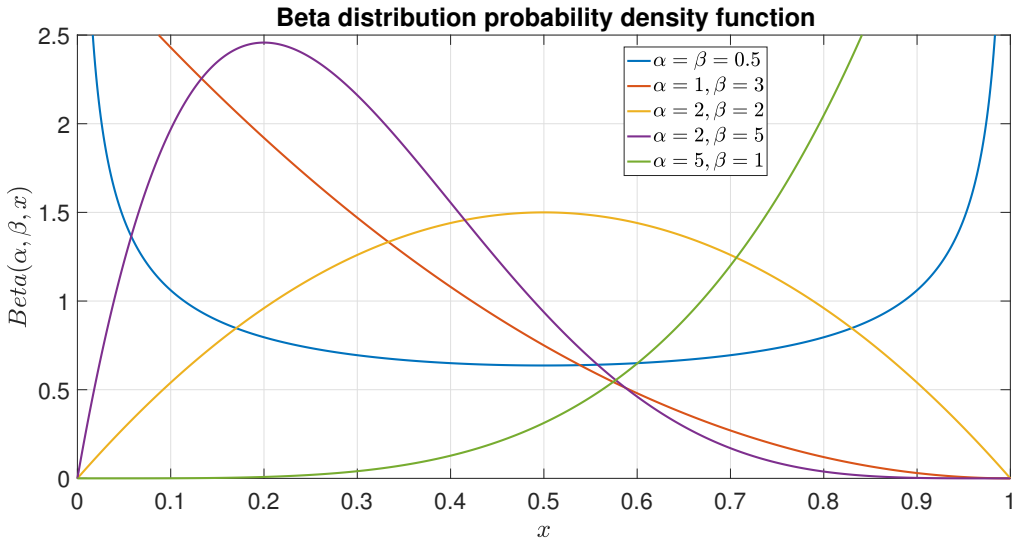

Figure 5.9: Probability density function of Beta distribution for different parameters.

the connections **without** "isink" and the connections which have less than 1000 sinks. Namely, the green dots in Figure 5.8. Some fits seem to be not very accurate. This is an indication that there might be a more appropriate family of distributions to describe the statistics of the data. To make sure the data set is rich enough, we combine data sets `place_25`, `place_50` and `place_75` of the data set 2. We call this data set training data. The histogram and the resulted fitting curves are depicted in Figure 5.10. The parameter values of α and β are shown in Table 5.3.

Table 5.3: Values of α and β for the training data.

Manhattan distance	$d_M \in [0, 5)$	$d_M \in [5, 10)$	$d_M \in [10, 15)$	$d_M \in [15, 20)$
α	6.2233	5.0182	4.9405	5.2435
β	397.9271	228.9191	188.9371	176.3081
Manhattan distance	$d_M \in [20, 25)$	$d_M \in [25, 30)$	$d_M \in [30, 35)$	$d_M \in [35, 40)$
α	5.3851	5.2903	5.7696	5.5687
β	164.3603	148.3772	147.8672	134.2422
Manhattan distance	$d_M \in [40, 60)$	$d_M \in [60, 8)$	$d_M \in [80, 120)$	
α	5.9534	5.9899	5.9899	
β	130.6464	112.7353	112.7353	

To test whether the obtained probability density function can be used to approximate the distribution of the remaining data in data set 2, i.e., the data in `place_100` and `place_150`, we plot the histogram of the data in `place_100` and `place_150` and the

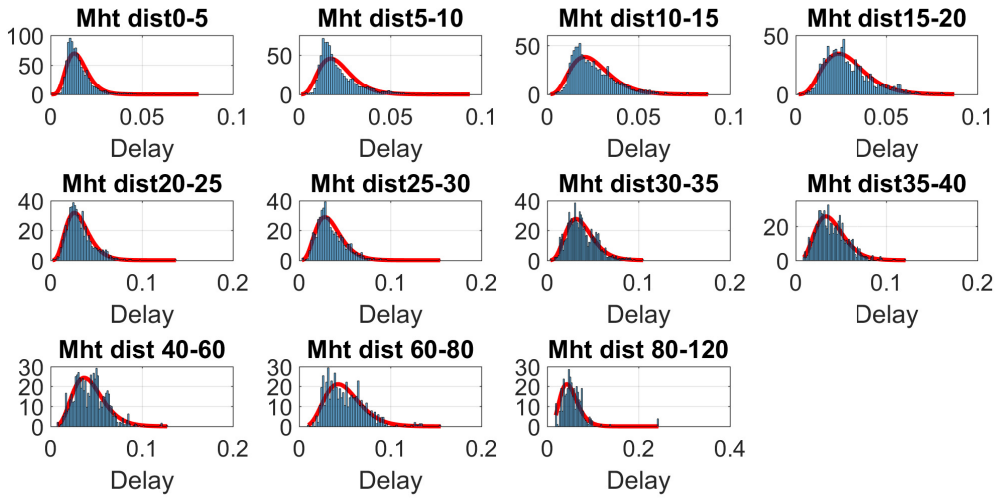

Figure 5.10: Histogram and fitted probability density function curves for `place_25`*,* `place_50` *and* `place_75` *in data set 2.*

probability density function curves. The results are shown in Figure 5.11. It can be

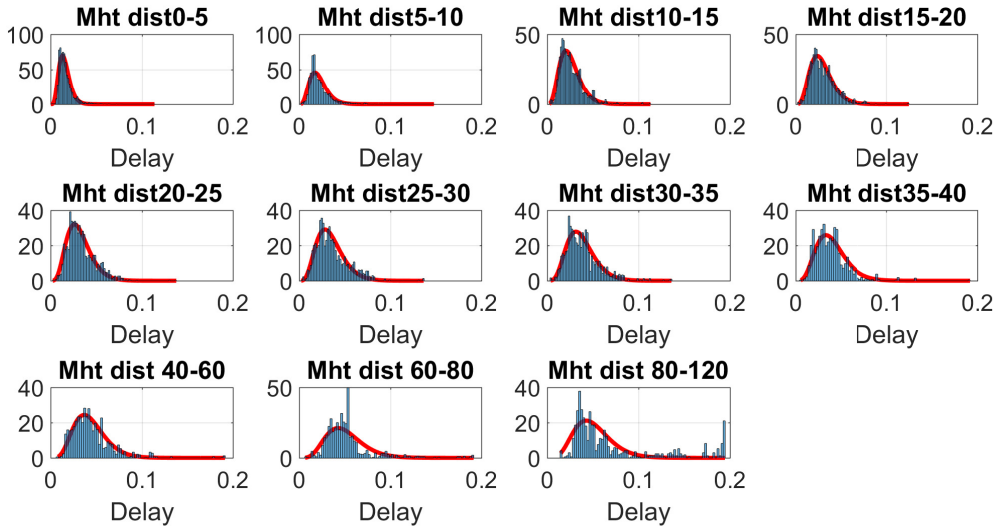

Figure 5.11: Histogram of the delay data in place 100 and 150 and the probability density function curves derived from the training set.

seen that the probability density function curves can still fit the histogram of the data in `place_100` and `place_150` very well when the Manhattan distance is less than 60. When the Manhattan distance is between 60 and 120, the differences are much larger.

This is most probably caused by the lack of data point when $d_M \in [60, 120)$. Hence, the statistical analysis approach requires to have enough data.

When the probability density functions are approximated, we still need to find a relation between the Manhattan distance and the delay. One possibility is to simply use the mean and variance calculated from the Beta distribution and then connect the mean to make a piece-wise linear function to approximate this relation. However, due to the time constraints, we could not further pursue this direction. In the following, we investigate the mean-bound method (introduced in the next subsection), which has the same reasoning.

For a more complicated model, which may provide a better approximation of the relation between the Manhattan distance and the delay, Gaussian process regression (see, e.g., [3]) can be a potential future research direction.

5.5.2 The AM-method

In this section, we explain how to obtain a candidate for the delay function by taking either averages or medians of the data points. The abbreviation "AM" refers to average and median. First, we illustrate the method by starting from the data set as represented in Figure 5.8, where as explained in Section 5.5.1, we only consider the connections without 'isink'. Secondly, we motivate the reasoning behind the AM-method.

The algorithm basically follows three steps:

1. First, divide the domain of Manhattan distances of Figure 5.8 into subintervals I_i, $i = 1, \ldots, N$. The length of each subinterval I_i and the number of subintervals N are parameters that have to be chosen.

2. Secondly, for each subinterval I_i, determine the average delay time t^i_{av} from the data points contained in $I_i \times [0, \infty)$. Alternatively, one can take the medians.

3. Thirdly, denoting by x_i the left-point of the subinterval I_i, connect for each $i = 1, \ldots, N-1$ the data points (x_i, t^i_{av}) and $(x_{i+1}, t^{i+1}_{\text{av}})$ by a linear function.

Figure 5.12 shows the resulting delay function when following the above procedure. When comparing the delay-functions from Figure 5.12 to the test data, we obtain a root-mean-square error of 0.01215 for the "average"-delay-function, and a root-mean-square error of 0.01208 for the "median"-delay-function. **Both of them outperform the state-of-the-art method used by Synopsys.**

The motivation behind the AM-method is similar to the one given in Section 5.5.1. The basic assumption is that the distribution of delay depending on the Manhattan

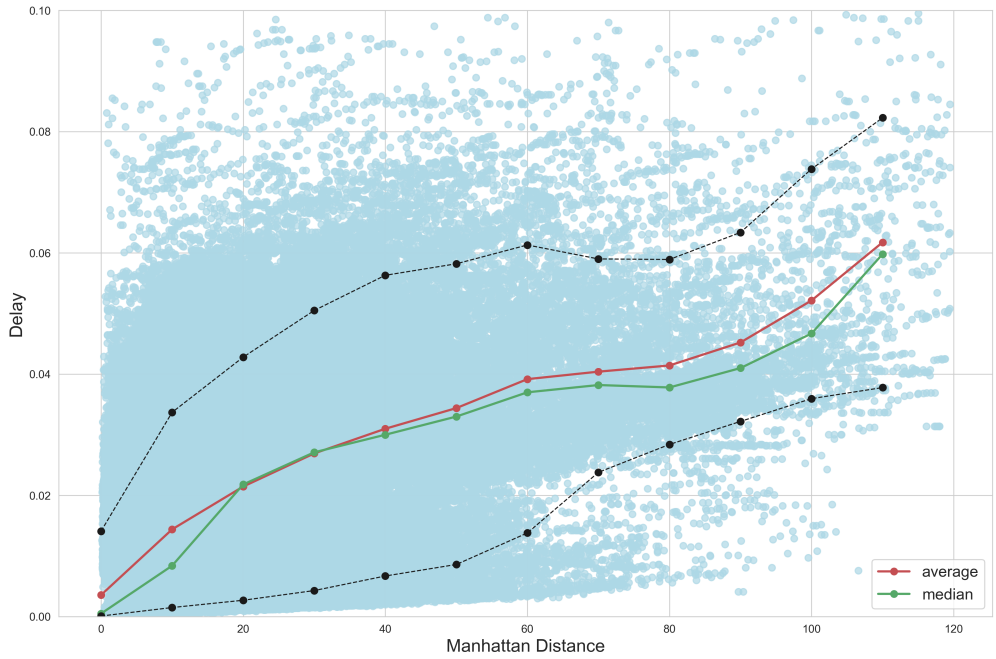

Figure 5.12: The delay functions obtained by the AM-method. The delay function in red is obtained when taking averages, the one in green when taking medians. The black curves give an idea of the standard deviations of the distributions in the corresponding category.

distance is meaningful. If that is the case, then a statistical analysis of these distributions might improve the estimate of the delay function. The difference of the AM-method to the distribution-fitting method of Section 5.5.1 is that the averages or medians are computed directly from the data, rather than being derived from distributions.

5.6 M4: Incorporate the connection structure

It seems that in several of the approaches proposed we are not utilising the network/graph structure. We know that there are several connections, regions of the IC are dense, and other areas are sparse, and this is not being incorporated in the predictions.

5.6.1 Speed of signal propagation

For convenience let's assume that the signal delay is approximately proportional to the distance between the driver and the sink. (This is in contrast to the theoretical/physical

analysis based on a wire's capacitance which predicts quadratic dependence, but the data seems to show approximately linear behaviour). In a manner similar to most introductions to variational calculus, we can consider there as being some speed profile (field) for signals propagating across the IC. If we suppose that the speed of signal propagation is homogeneous and isotropic, then the quickest path is a straight line between any two points. (There is the constraint that wires are confined to Manhattan paths along the circuit, but if most connections are locally small then a straight line path is likely not an awful approximation). We are interested in the related problem, which appears to be the reverse of this, namely, that if we are forced to travel in an approximately straight line, can we predict the signal's travel time. The key ingredient here is then the construction of the signal's speed profile, which is an area where we suspect the network structure of the connections can be exploited (hopefully intelligently).

5.6.1.1 Incorporating the network structure

Consider the following example dynamics, which is that when several drivers are closely packed together in a dense region, that the area appears quite congested. Perhaps such a scenario would suggest that there is a large amount of signal traffic in this area, and hence that small perturbations of a driver/sink in this region will drastically influence the signal delay.

Perhaps the converse might be true, and that a high density of drivers/sinks in a region is indicative that the area is a popular and fast flowing region, and that signals travel fast in this area. Hence perturbations here only produce small changes in the delay time.

Another possibility is that if we were to draw in all the paths the signals travel along, there are regions where these overlap extensively, and then sparser regions. In this case it is the degree of overlap that a connecting path would need to travel through which would determine the signal delay.

5.6.1.2 Constructing a speed profile

For the sake of convenience, let's suppose that the density/clustering of drivers in a region determines the signal's travel time. There are a few means by which we can define some form of density profile. One which comes to mind is to construct a *kernel density estimate* (KDE), where the neighbourhood on the IC which a driver congests is reflected in the choice of bandwidth. An example of one such KDE is shown in Figure 5.13.

One of the pitfalls of a KDE is that it is expensive to evaluate. However, if speed is

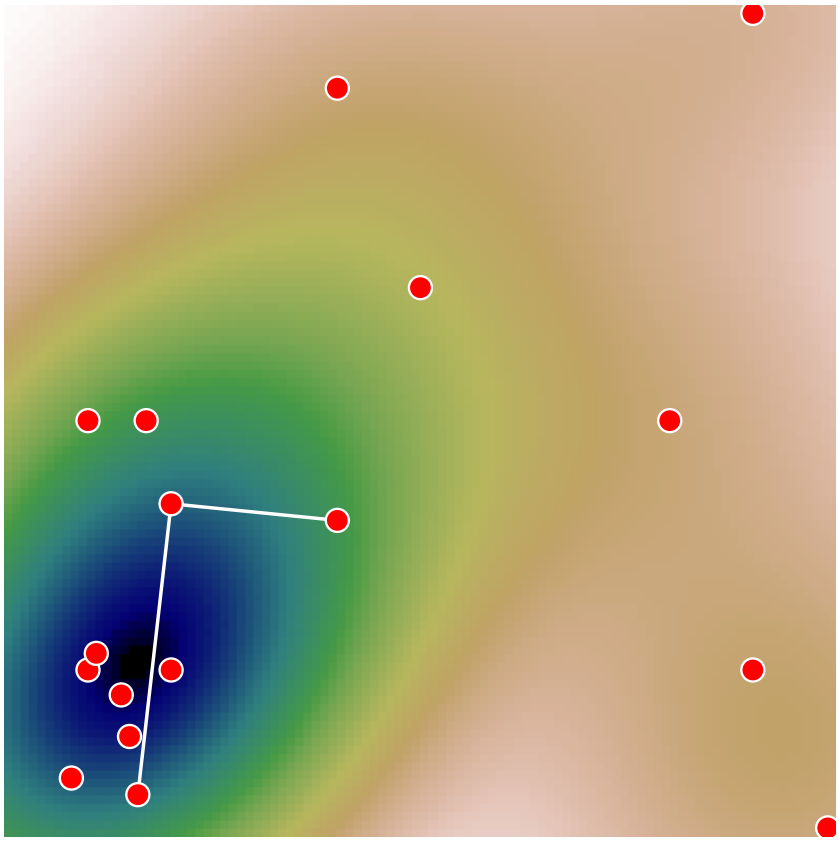

Figure 5.13: An example of a kernel density estimate constructed for a selection of drivers. We highlight two example connections, where one passes through the most dense region, and another along a less dense region.

very critical then perhaps this can be approximated by some sufficient order 2-dimensional polynomial. The advantage of this is that then the integrated travel time is trivially evaluated computationally (requiring no computationally intensive approximate integration techniques). A second pitfall is that the KDE's range is $[0, \infty)$, where in reality we would want some baseline speed. This can be achieved by some transformation of the form

$$\text{speed} = \alpha + \beta \times \text{density}, \tag{5.7}$$

where α is strictly positive. The exact form of these coefficients, and also the mapping could be explored during a calibration/training stage.

Overall, once some speed profile $v(\boldsymbol{r})$ has been constructed, and two points A and B have been found, then we approximate the delay t_D as

$$t_D \approx \int_{\boldsymbol{r}_A}^{\boldsymbol{r}_B} v(\boldsymbol{r})\, d\boldsymbol{r}\,, \tag{5.8}$$

where the integral is trivially evaluated if polynomial approximations (or splines, Cheby-

shev functions, etc.) are used. Furthermore, this is trivially differentiated or evaluated under small perturbations.

5.6.2 Other network/graph features

So far this has used a very geometric interpretation of the network. However, it is not too complicated to also pull out prediction features, such as degree of connectedness, the number of vertices forming a cliques, etc., then these can also be used as features (possibly). Other simpler degrees of how crowded a region is would be its nearest neighbour distance, or the number of neighbours within some region ε around a node. Other ideas from graph theory might be to try and classify regions that might form a *small world network*, and see if the delays of connections within these regions show a structure/correlation different to large world networks.

5.7 M5: Data-based Prediction via Gradient-boosting regression

5.7.1 Problem Setup

In this section, a pure data-based delay predictor will be presented based on gradient-boosting regression (GBR). Generally the available data for prediction are associated to the driver and the sink, respectively. The name and label of each data is not clearly related to the delay, which can be ignored in later discussions. Other data regarding the driver and sink (isink) are described as follows.

- driver: position (X_d, Y_d), rise r_d, fall f_d, sink driving number N_d.
- sink/isink: position (X_s, Y_s), rise r_s, fall f_s, slack S_k.

From explanations from the company engineers, the delay can be described as the rise difference $r_s - r_d$ between driver and sink.

From the current experience in Synopsis, the delay is influenced by the distance between drivers and sinks, and not related to respective positions, leading to the initial need to find a mapping from the distance to the delay (which is described in the competition description).

However during our discussions and from the analysis in Section 5.3, the sink driving number N_d is found to have a great influence for the delay. This leads to our predictor design to accept input of distance and the driving number simultaneously in the following.

More specifically, the delay prediction task is set to find a mapping from three inputs (X-distance,Y-distance,driving number) to the delay estimate of $r_s - r_d$, say P_d,

$$\varphi : (|X_d - X_s|, |Y_d - Y_s|, N_d) \to P_d. \tag{5.9}$$

We have tried different prediction models to fit current historical data where the gradient-boosting regression (GBR) model outperforms in our limited tests. Hence the following discussion contributes to introduce GBR models and its empirical performance on the training data provided during the SWI workshop.

5.7.2 Gradient-boosting Regression

Gradient-boosting (see [2]) is a powerful algorithm widely used in data-driven machine learning tasks like regression, classification and ranking. It belongs to a general ensemble learning algorithm called boosting methods, where base estimators are built sequentially such that combined estimators' bias can be reduced. The target of GBR is to learn a strong model from the combination of several weak models.

The basic idea of boosting starts from following setup:

- A loss function, say $L(y, f(x))$ for the true y and the prediction $f(x)$.
- Weak learners, say $h_i, i = 1, ...$
- An additive model to add weak learners to minimise the loss function, say $F(x) = \sum_{i=1}^{M} \gamma_i h_i(x)$ for pending coefficients γ_i.

Specifically in the Scikit-learn package, the weak learners are chosen as decision trees. From the official help document of GBR model, "Gradient Tree Boosting uses decision trees of fixed size as weak learners. Decision trees have a number of abilities that make them valuable for boosting, namely the ability to handle data of mixed type and the ability to model complex functions."

Moreover with the tree learners, the additive model itself can be formed as follows to fit the sequential learning,

$$F_i(x) = F_{i-1}(x) + \gamma_i h_i(x), \tag{5.10}$$

where the weak learner h_i may come from the following optimisation problem:

$$h_i = \arg\min_h \sum_{j=1}^{n} L(y_j, F_{i-1}(x_j) + h(x_j)). \tag{5.11}$$

Here $\{x_j\}_{j=1}^{n}$ denote the train data samples.

To solve the above optimisation problem is not facile, and an insightful understanding of gradient-boosting is to introduce the gradient-descent algorithm in the above boosting setup. Hence a GBR model is finally described as an updating algorithm as follows,

$$F_i(x) = F_{i-1}(x) - \gamma_i \sum_{j=1}^{n} \nabla_F L(y_j, F_{i-1}(x_j)), \tag{5.12}$$

where γ_i can be chosen by linear search with a constant $\gamma > 0$,

$$\gamma_i = \arg\min_{\gamma} \sum_{j=1}^{n} L(y_j, F_{i-1}(x_j) - \gamma \frac{\partial L(y_j, F_{i-1}(x_j))}{\partial F_{i-1}(x_j)}) \tag{5.13}$$

5.7.3 Empirical Results

To design the predictor, we will use the gradient-boosting regression models in Scikit-learn package directly and our experiments are done in a laptop workstation with i7-7700HQ CPU.

GBR model is setup with 100 estimators and maximal search depth of 7, which is trained and tested in data-set-1 and data-set-2 respectively. Following the general setting in the group discussion, we use the place-150 data as train data, and the place-100 data as test data in both data-sets. The test results measured by RMSE is illustrated in Table 5.4, which achieves the best test results in all of our current trials (M1-M5). Also the testing performance is illustrated in Fig. 5.14 and 5.15 respectively.

The accuracy of GBR has a higher time-cost. From our experiments, the GBR model is about 70 times slower than the linear regression model (0.02961s v.s. 0.000475s) based on the testing time over 12081 samples for data-set-1; about 200 times slower than the linear regression model (0.5184s v.s. 0.002507s) based on the testing time over 191440 samples for data-set-2.

Metric	Data-set-1:place-100(sink)	Data-set-2:place-100(sink)
RMSE	**0.03845777**	**0.00663324**

Table 5.4: Test results on chosen data-sets. Models are trained from the place-150 data in data-set-1 and data-set-2 respectively.

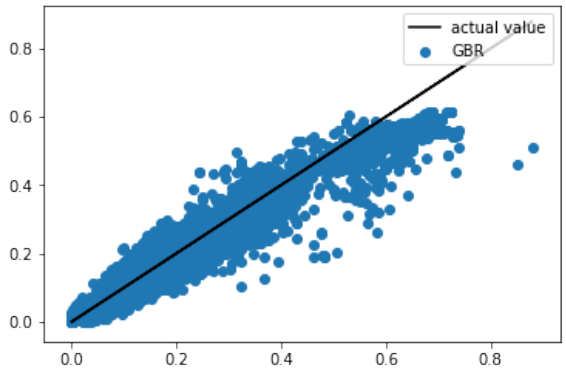

Figure 5.14: Test on Data-set-1:place-100

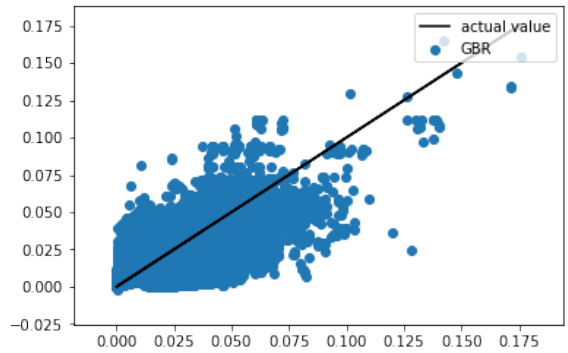

Figure 5.15: Test on Data-set-2:place-100

5.8 Recommendations

From our experiments the most performing and robust method turned out to be **M3: Statistical method**. At this point we recommend the implementation of this method. It is possible that upon further fine-tuning of the optimiser and classifier other methodologies turn out to be superior. Regardless the method chosen, benefits can be gained from a classification before the data is fed to the optimiser. In this regards, we can formulate the following *recommendations*:

- do **not** use the libcell label for classification. This does not lead to a meaningful subdivision of the data in the sense that each resulting subset exhibits trends not already present in the complete set.
- provide more detailed information about the component type. It is possible that this will lead to a meaningful classification with predictive power as pertains delay estimation. Examples of parameters we imagine could provide more insight are: components size, component power, and number of ports.
- **do** use sink vs isink for classification. From our analysis it is clear that components at the end of lines containing odd/even numbers of invertors exhibit significantly different trends and statistics. We expect this not just to be a property of the parity, but of the exact number of invertors. Having access to this datum likely results in more accurate predictions.
- **do** use the number of sinks a driver powers for classification. This information is partially available from the data files provided.
- **do** use the location of sinks, combined with the local sink density for classification. *In our tests this was not considered due to time restraints.*

5.9 Conclusion

An improvement over the state-of-the-art has been observed. It is however likely that a stronger correlation, and thus a better delay estimator, can be created by first pre-processing the data given. It is possible e.g., that distance to the boundary of the chip, crowding and etc., significantly affect the final delay after routing.

A good place to start in our opinion for further improvement, would be to study the routing algorithm and try to extract heuristics and patters from its outcome and inner workings. This can help to direct the search for strong correlations in the placement vs delay function.

It is unclear at this stage what the optimal accuracy of any delay estimator is. Obviously a perfect delay estimator requires the consolidation of placement and routing; this is prohibitively costly and should not be aimed at.

Another question is how robust any solution is in transitioning from one generation of micro-architecture to the next. Can the delay function be copied over? This is unlikely, but it seems feasible that the type of considerations that lead to the delay function remain valid across generations.

Finally we would like to thank SynopSys and its representatives for this opportunity. It is interesting to see what wealth of technology and know-how lies behind a truly omnipresent technology. This introduction has at the same time an effect of demystification and deep appreciation of the designers and design methodologies!

Acknowledgements

We would like to take the opportunity to explicitly thank the NWO for their support of the SWI initiative and the delegates from Synopsys for their time and help during the week. It was an enriching experience and a good showcase of how mathematics can play an important role in the industrial design process. Also thanks to the reviewer to take the time to go through this document and to help improve it.

Bibliography

[1] Gamma function Wiki Page

[2] Gradient Boosting Wiki Page

[3] Rasmussen, C.E. and Williams, C.K.I., Gaussian processes for machine learning, MIT Press, 2006