國立臺灣大學電機資訊學院電子工程學研究所

碩士論文

Graduate Institute of Electronics Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

針對多電子束直寫系統資料壓縮比例之細部繞線

Data Compression Ratio-aware Detailed Routing for Multiple E-Beam Direct Write Systems

邱煜翔

Yu-Hsiang Chiu

指導教授：陳中平 博士

Advisor: Chung-Ping (Charlie) Chen, Ph.D.

中華民國 104 年 9 月

September, 2015

國立臺灣大學碩士學位論文
口試委員會審定書

針對多電子束直寫系統資料壓縮比例之細部繞線
Data Compression Ratio-aware Detailed Routing for Multiple E-Beam Direct Write Systems

本論文係邱煜翔君（R02943148）在國立臺灣大學電子工程學研究所完成之碩士學位論文，於民國 104 年 09 月 24 日承下列考試委員審查通過及口試及格，特此證明

口試委員：

系主任、所長

誌謝

終於輪到我寫謝詞啦！

首先要感謝我的父母家人，從小到大總是無怨無悔當我最堅實的後盾，讓我沒有絲毫後顧之憂，如果沒有你們絕對不會有今天的我。

感謝我的指導教授陳中平老師，讓我有了這樣的環境可以與同儕砥礪戮力研究。感謝方姊姊在我的研究生涯迷茫無方向時及時出現，並且當做自己的學生一般指導。感謝郝市長隨時願意提供你的專業知識，並且在生活上也給了我非常多的建議。

實驗室的大家都很善良，大白不只是一個好同學也是一個好室友，希望日後還能有機會一起找家賢哥繼續嘴砲泰宇。阿智很久沒出現了，但吃飯想揪人的時候還是會讓人想起當年跟 Bonnie 一起修課的日子。家豪哥和紅線老大指導了我不少東西，力巴哥跟孝銓哥則是嘴砲時的好夥伴。感謝博班的學長們，特別是士倫學長，對我的厚愛與教導。昶毅、砲灰、謝等等 ICS 組的學長同學們也陪伴了我度過無數歡樂與苦悶的時光。至於學弟妹們我就不一一點名了，希望你們可以在這間實驗室找到自己的研究和人生的道路方向。此外更希望的是我沒有漏掉哪一個重要的人。

感謝辜狗哥三不五時提醒我畢業時程中所剩下的時日無多，雖然很煩但也是很重要的啦。感謝小枝阿台常常陪我吃飯，並且在求學及求職過程中給了我無數重要的建議。感謝薄薄小黑在我碩一的時候讓我除了自己的實驗室以外還有別人的可以去鬼混。本來還想要感謝一下 Keyway 的，但想了半天實在沒想到這兩年多來你有為我做甚麼事，就感謝你總是當我們的話題守護者吧。其他想要感謝的以往的同學朋友們實在太多了，一言以蔽之就感謝國小同學國中同學高中同學和大學同學們吧！

最後壓軸的是品嘉，感謝妳在我的學生生涯尾端加入我的人生。因為有妳在我身邊，我覺得我甚麼事情都可以辦得到。雖然這本論文妳真正參與的份量約莫只是三分之二，但我想以後還是會有很多需要麻煩妳的地方。妳很好，各個方面都是，對我而言很重要，謝謝妳。

好，我要畢業了！

中文摘要

由於製程的演進，超大型積體電路的最小關鍵尺寸已趨近於物理極限，而傳統光學曝光所使用之光源因其解析度而逐漸不敷使用，電子束曝光則因其高度的精準特性而成為極具潛力的次世代製程選擇。

電子束的精準程度可以達到奈米量級，在使用上必須非常精確的將電路資訊傳輸至曝光系統。而現今的超大型積體電路複雜程度與日俱增，製程上若欲讓電子束機臺得以即時曝光顯影生產，就必須仰賴極有效率的資料傳輸方式，將電路資料即時傳輸至機臺上，此傳輸規格超越了現今光纖傳輸所能達到的極限。因此實際在工業上的使用，必須先將電路的資訊壓縮以後再傳輸，至機臺上解壓縮，才能達到預期的產率。

本篇論文將要探討的問題是，若已經選擇了特定的壓縮演算法，是否能夠在電路實體設計的階段，就產生出能夠讓此壓縮演算法表現得更加優異的電路布局，進而提升整體的壓縮效率。而實驗的結果證明了此一理論，也同時說明了由繞線階段便加以考量，進而影響資料壓縮的效果是不容忽視的。此一領域亦極具發展的潛力與研究價值。

關鍵字：電子束曝光、資料壓縮、實體設計、電路布局、繞線

ABSTRACT

The feature size of Integrated Circuits(IC) are shrinking down along with the advancement of technology, but the resolution of the ArF laser is far from the target for next generation lithography. Electron beam (E-beam) lithography, with its high-accuracy characteristic, is very likely to become the main role in next generation lithography.

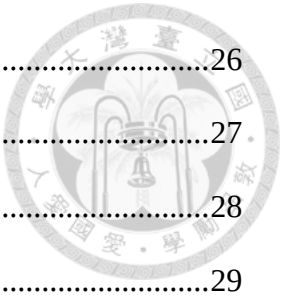
Because of the accuracy of E-beam, the exact information of the circuit has to be delivered to the E-beam emitter. However, circuits nowadays has become so complicated that the successfulness of this process relies on the speed of data transmission, which is not sufficiently fast even with technologies today. So in practice, data should be compressed first, transmitted by optic fibers, and then decompressed in the E-beam machines.

In this thesis, we proposed a detailed routing method to improve data compression quality before applying the actual compression algorithm. The results of experiments show that, with one particular data compression algorithm, LineDiff Entropy, chosen, we improve data compression ratio with our proposed detailed router. And we can conclude that considering data compression ratio in physical design phase is a field worth studying.

Keywords: Lithography, Electron Beam, Data Compression Algorithm, Physical Design, Detailed Routing

CONTENTS

LIST OF FIGURES



LIST OF TABLES

Chapter 1 Introduction

As long as the rapid advancement of technology today, the size of integrated circuits (IC) are getting smaller and smaller these days. However, diffraction is now a very severe problem due to the limitation of 193-nm light source in traditional lithography [1]. So we have to search for an alternative way with high resolution, but throughput still comparable to today's optical lithography systems.

Among all candidates, Electron Beam Lithography (EBL) systems are well known to produce excellent resolution, good line edge roughness and good line width roughness [2], but they also have a common problem in their low throughput. A simple way to improve it is to use the concept of massive parallelism [3]. So a Multiple Electron Beam Lithography (MEBL) system should be a feasible alternative.

Another benefit of using MEBL is that, due to its high precision, we can now apply a maskless process into practice. The traditional optical projection systems use a mask to project the entire chip pattern; while a maskless system, also known as a "direct-write" system, allowed us to use electron beams to draw custom shapes directly. The concept is that there is an electron-sensitive film on the surface called a resist. And while exposure, the beams change the solubility of the resist, results in the removal of either the exposed or non-exposed regions after immersed in a solvent. The most significant advantage of a Multiple Electron Beam Direct-Write (MEBDW) system is that we can easily modify the image we want to project by changing the dose of each emitter, whereas a mask once made is difficult to modify [4]. And with a direct-write system we can also save our expenses of the masks.

There already exist some applications of MEBDW systems, Figure 1.1 shows a concept of Reflective Electron Beam Lithography (REBL). It's an industrial application

of MEBDW system by KLA-Tencor Corporation. Within this thesis, we'll focus on the feasibility of this application and try to make it more practical and play its role in next generation manufacturing process.

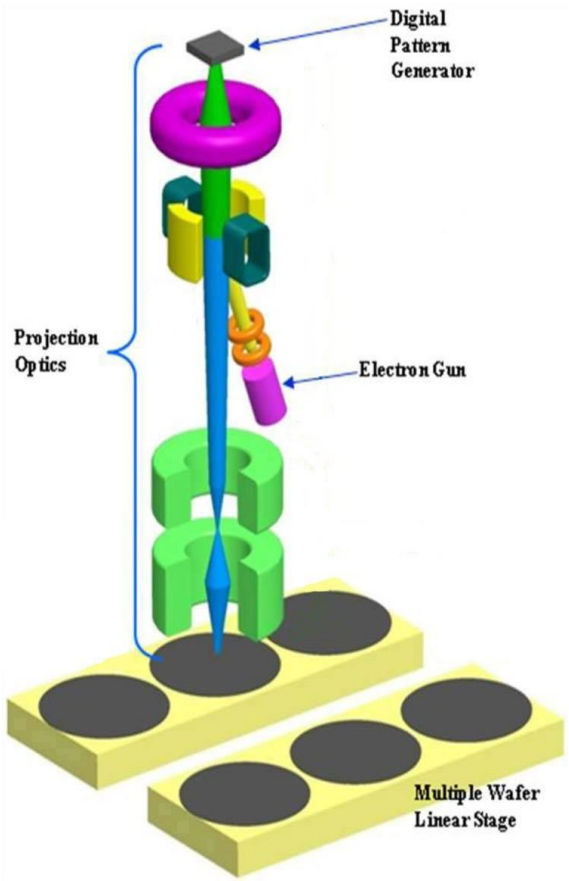


Figure 1.1 Concept of REBL nanowriter. [5]

1.1 MEBDW Systems Difficulty

Every advantage has its disadvantage, there also exist some problems in MEBDW systems. One of them, maybe the most important one, must be the limitation in data transmission rate.

Because of the high accuracy of electron beams, we have to deliver the exact circuit information called a "bitmap" to the electron beam emitters. Bitmap is a data type of circuit layout. And just like a monitor, it displays information pixel by pixel.

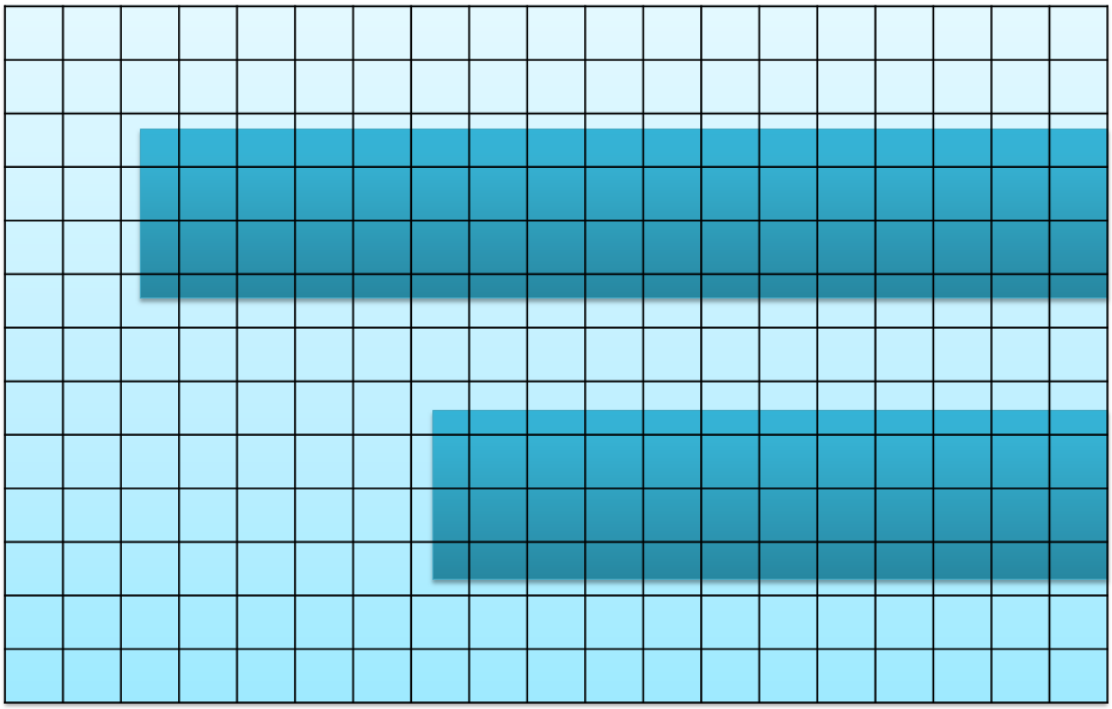


Figure 1.2 Circuit layout example.

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	14	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
00	00	18	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31
00	00	18	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31
00	00	12	09	09	09	09	09	09	09	09	09	09	09	09	09	09	09	09
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	12	18	18	18	18	18	18	18	18	18	18	18
00	00	00	00	00	00	00	16	31	31	31	31	31	31	31	31	31	31	31
00	00	00	00	00	00	00	16	31	31	31	31	31	31	31	31	31	31	31
00	00	00	00	00	00	00	14	12	12	12	12	12	12	12	12	12	12	12
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Figure 1.3 5-bit bitmap transformed from Figure 1.2.

Figure 1.2 and Figure 1.3 show an example of bitmap transforming. A pixel represented by 00 in bitmap shows there's no content in the pixel, while 31 shows that the whole pixel was occupied. And any other number in between shows the ratio of area occupied to total area in that pixel.

Device Specification		Maskless Process Specification	
Wafer size	300 mm	Pixel size	7 nm * 7 nm
Writing rate	70 WPH	Pixel depth	5 bits
Writing time	50 s	Wafer data size	7200 Tb
Optical fiber transmission rate	10 Gbps	Required rate	144 Tbps
Number of fibers connected	32	Required fibers	14400
450 times exceeded			

Table 1.1 Specification of data transmission rate.

And in Table 1.1, we can now estimate the data volume and required data transmission rate. From [6] we can get our device specifications shown below: wafer size is 300 mm in the diameter and expected throughput is 70 wafers per hour (WPH), which gives us about 50 seconds to write a single layer of a wafer. The digital pattern generator (DPG) in a REBL system is a chip of less than 26 mm * 33 mm, so the number of optical fibers connected to DPG is limited. And we can also know the data transmission rate of a single fiber is about 10 G-bits per second (Gbps) [1], [3].

On the other hand, let's take a look at the specifications of MEBDW systems. The pixel size should be defined by half the minimum feature size [5], so we set our pixel size to be 7 nm each side to match-up the 14-nm technology node. And we'll use a 5-bit gray level in each pixel, which gives it 32 different levels. Total data size can be

estimated by equation 1.1.

$$Data\ size = \frac{Area\ of\ wafer}{pixel\ size} \times pixel\ depth = \frac{\pi \times 150 \times 150 mm^2}{7 \times 7 nm^2} \times 5\ bits \approx 7200\ Tb \quad \text{..... (1.1)}$$

So now we can know that our required data transmission rate is 7200 / 50 = 144 T-bits per second (Tbps), which required 14400 optical fibers operating at the speed of 10 Gbps to work simultaneously. That's about 450 times over the acceptable number of fibers today. We definitely need to do something more to make it work.

There are several different architectures to build up MEBDW systems, which we'll give a brief introduction in the following chapter. But to fit the data transmission limitation we just discussed, the procedures in Figure 1.4 is the only adequate solution. The bitmap file has a very huge volume, so in practice we have to compress it first, and then transmit the compressed file instead. In the end we decompress it just before lithography.

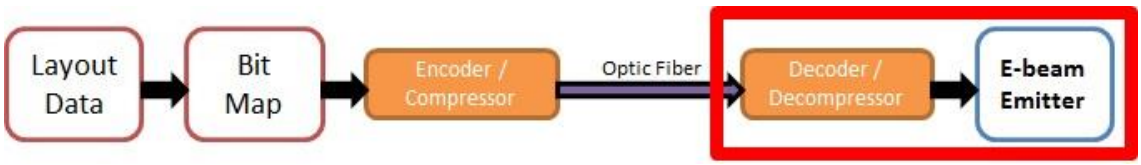


Figure 1.4 Procedures in MEBDW system.

1.2 Motivation and Accomplishment

Just like we've talked about in section 1.1, we need a data compression algorithm in MEBDW systems. And among all such algorithms, two major characteristics we have to pay attention to are respectively data compression ratio and decompression rate. Data compression ratio directly affects the feasibility of MEBDW systems, so it's definitely

worthy a first priority concern. Decompression units are placed by the electron beam controllers just like in Figure 1.4, and that's why we don't want to put too much hardware resource onto it but still want the emitters to work in real time. So decompression speed is also an important matter.

There do exist many data compression algorithms, some of them are even originally designed to compress bitmap data. But what we are really interested in is that whether we can improve the results of compression even before we get to that part. We want to enhance the quality of data compression in physical design phase of the circuit. In other words, with a specific data compression algorithm selected, we want to design a router that generates layout patterns performing better in such algorithm. And so far, there is no published material shows something like this research has been done. So we decided to do some research on this topic.

And the results of our experiments are pretty excited. A data compression algorithm that performs well in bitmap has been chosen. And with our proposed router, the data compression ratio can be further increased by up to 67% while decompression time remains unchanged.

But other than these numbers, the most valuable contribution of this research is that our conclusion can be further deductive to a whole new field: no matter what kind of data compression algorithm we fall to, we can always improve the performance by some modification in the physical design phase.

1.3 Organization

There are several theme topics will be illustrated in this thesis. We've talked about the introduction of the MEBDW systems and the problem we encountered in this

chapter. In Chapter 2, some of the existed architectures and algorithms related to our research will be presented. And our proposed detailed router will be specifically expressed in Chapter 3. And in the end, the results of experiments and our conclusion are respectively in Chapter 4 and Chapter 5. There will also be some discussion of what this work can be further enhanced in Chapter 5.

Chapter 2 Preliminaries

There are three major topics in this chapter. Section 2.1 will be a brief introduction of the MEBDW system architectures, which states the problem we confronted and why data compression algorithms are important. After that, some of the published data compression algorithms will be introduced in section 2.2, including the one performing extremely well with bitmap data, LineDiff Entropy. In the end of this chapter, an important routing algorithm and its predecessor will be given in section 2.3. All of the work in this thesis are developed mainly from all these three fields.

2.1 MEBDW System Architecture Designs

From Chapter 1, we are aware of how important MEBDW systems are. And speaking of the architectures of MEBDW systems, the most straight forward thought should be something like Figure 2.1. Many things should be a lot easier if we just connect the disk with layout data to the electron beam controller. But unfortunately, with our estimation in section 1.1, this kind of architecture requires a 144-Tbps transmission rate, which is much more than what we can accomplish today. So this is definitely not a solution.

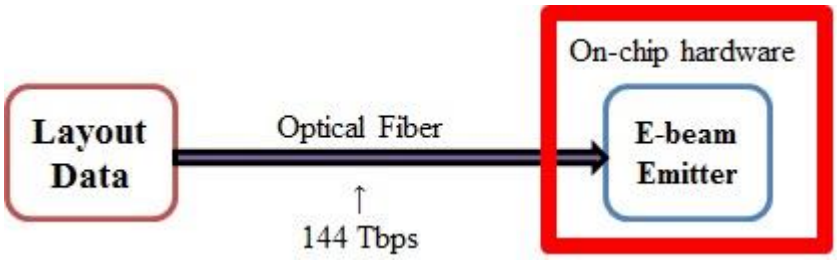


Figure 2.1 Direct transmission.

To avoid the data transmission rate limitation, another thought might be using memory. Just like Figure 2.2, we try to attach a memory onto the electron beam emitter. But if we want to store the information of whole wafer into the memory, we need a memory element of 7200 Tb according to the calculation before. Even if we don't need to keep all information of a wafer, instead we only store one chip, we still need a memory more than 20 Tb with a chip size 10 mm * 20 mm. But considering the small area of electron beam controller, memory size is estimated to be limited by 16 Gb [3] even with the highest-density DRAM.

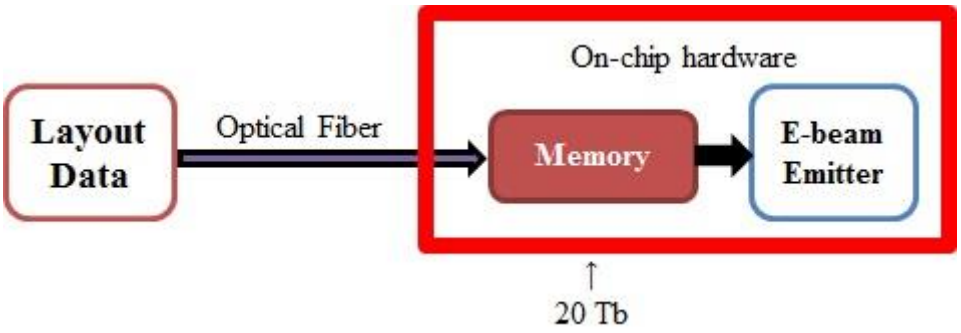


Figure 2.2 Whole chip information in on-chip memory.

It is probably the time to think about data compression now. The first architecture considering data compression might be like Figure 2.3. We compress the layout data and store them into the on-chip DRAM. And then our data needs to be decoded back to bitmap before sent to electron beam emitter. There are two difficulties in this architecture. The first one is the data compression algorithm. We've already known that the original data size of a single chip is 20 Tb and what we have is a 16 Gb DRAM, so our compression ratio (CR) must be more than 1250. That is almost impossible. Moreover, we'll need an on-chip decoder according to the data type. Considering the limited area, we actually can't expect to do these two things well at the same time.

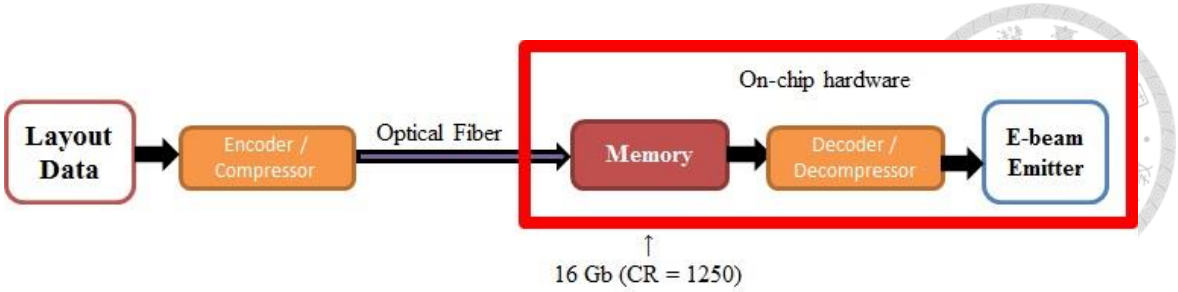


Figure 2.3 Compressed information in on-chip memory.

Now we know that we can't do too many things because of the area limitation on electron beam controllers, so let's try to move the memory and the decoder off-chip like Figure 2.4. But now we're still facing the problem of data transmission rate limitation in optical fibers.

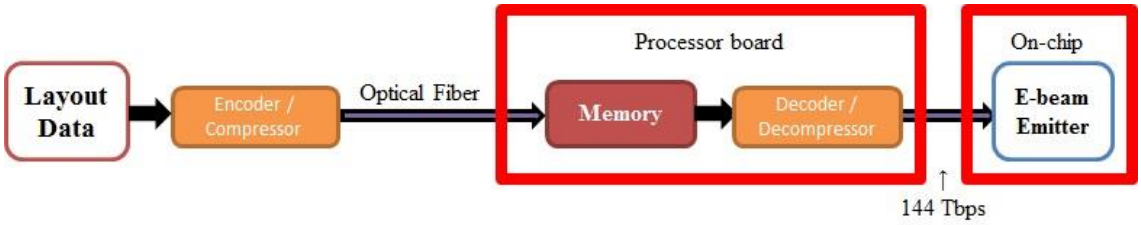


Figure 2.4 Off-chip memory and decoder.

Finally, we have found one feasible solution. Let the decoder on-chip, and all the other elements placed on the memory off-chip. If our data compression algorithm is capable of providing CR fitting the transmission limitation, and the speed of decompression can make the electron beam emitter work in real time, then it is good enough for us with architecture in Figure 2.5.

All the above are the architectures of MEBDW systems. Apparently there are not too many choices for us. Therefore Figure 2.5 is the only architecture we'll talk about in this thesis.

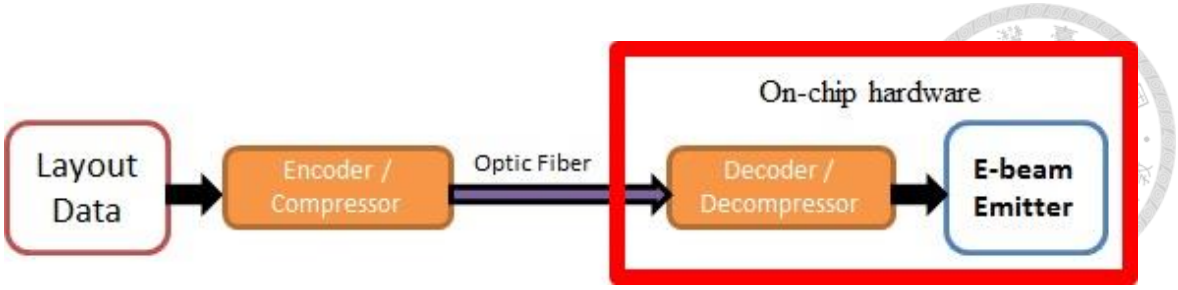


Figure 2.5 Architecture of data delivery in MEBDW systems.

2.2 Data Compression Algorithms

There are several published data compression algorithms now, and some of them were designed for images. Among those algorithms, some of them were even designed for data type like bitmap. LineDiff Entropy, proposed by Tang et al. from NTU in 2013 [1], is our favorite compression algorithm because it outperforms other algorithms in the aspects of both data compression ratio and decompression speed. So in this section, we're going to give a brief introduction of some data compression algorithms for bitmap, and a more-detailed one for LineDiff Entropy algorithm.

2.2.1 Introduction of data compression algorithms

First of all, LZ77 [7] is a lossless data compression method that was used in a variety of compression software including zip, gzip, WinZip and others [4]. The popularity of those software programs also reveals its significance. LZ77 is a dictionary-based compression algorithm that encodes raw data by a sliding window. Here is an example of LZ77 in Figure 2.6. The encryption format of each row is (P, L, c), where P is the offset between current character and the match one in the dictionary buffer, L is the length of the identical pairs, and c is the first character of the unencrypted stream. So we can conclude that LZ77 is an algorithm taking advantage of

the repetition of input data.

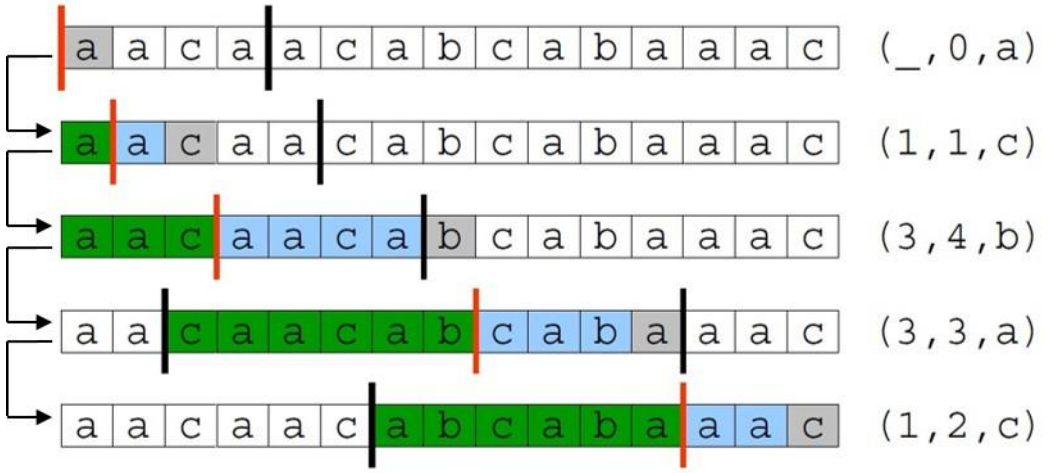


Figure 2.6 Example of LZ77.

LZ77 has been proved to be an efficient way to encrypt data in rows. But to apply this technique to bitmap, which is a flattened and rasterized data scheme, there's something more needs to be done. Figure 2.7 is an updated version of LZ77 called 2D-LZ [5] technique. The most significant difference between LZ77 and 2D-LZ is that we now have to memorize the coordinates of the repeated pattern because we work on patterns in a 2D-plane now.

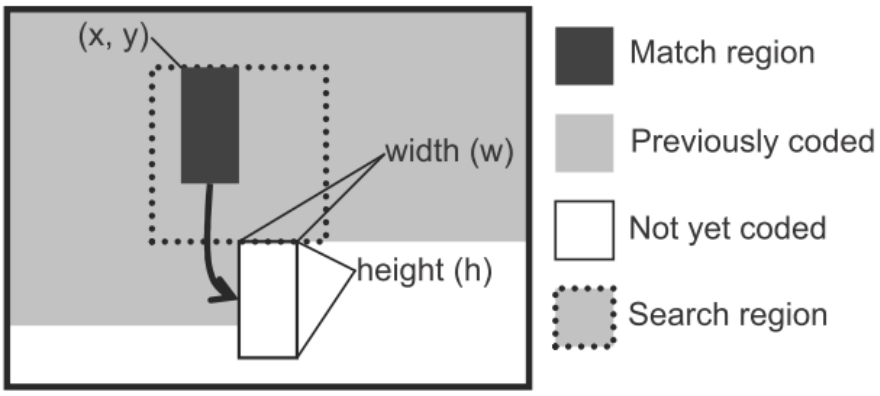


Figure 2.7 Example if 2D-LZ. [5]

The compression techniques in LZ is based on the highly-repetitive characteristic of the data map, but what if the layout is not so repetitive? Another widely-used way is to compress data by context-based prediction schemes. Figure 2.8 shows two example of such prediction.

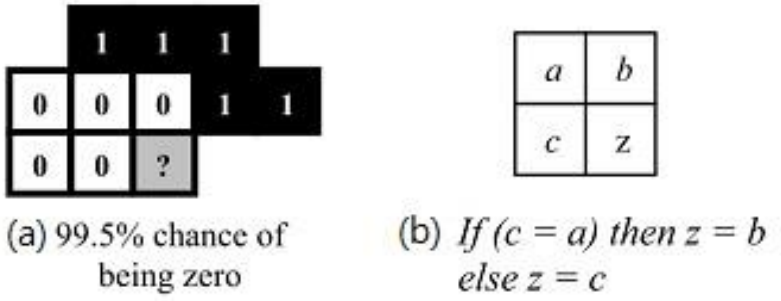
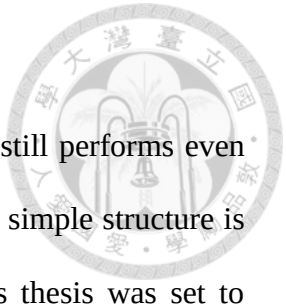


Figure 2.8 Context-based prediction:

(a) ten-block prediction (b) three-block prediction. [4]

Joint Bi-level Image experts Group (JBIG) is a standard for lossless compression of binary images, developed jointly by the CCITT and ISO international standards bodies. JBIG uses a 10-pixel context to estimate the probability of the next pixel being white or black [5], just like shown in Figure 2.8(a).

To combine the advantages of the LZ-style and context-based prediction, Vito Dai from University of California, Berkeley, has proposed a well-known compression algorithm called Context-Copy-Combinatorial Coding (C4) and its advanced version called Block C4 in 2008 [4]. Both of these two algorithms accomplished this goal through automatic segmentation of an image into copy regions and prediction regions. And data in copy regions can be easily dealt with LZ-style encryption, while three-block prediction format (Figure 2.8(b)) is used in the prediction regions. So these algorithms performs extremely well when it comes to bitmap data compression.



2.2.2 LineDiff Entropy

Despite the high efficiency of Block C4, LineDiff Entropy [1] still performs even much better in bitmap data compression and decompression. And its simple structure is also elegantly attractive. So the main target of the research in this thesis was set to improve the quality of this algorithm. That is also why we'll give a comprehensive presentation of this algorithm in this section.

LineDiff Entropy is a data compression algorithm designed for pixelized data format like bitmap. It was designed to access files with 1024 pixels in the width and 5-bit gray level for each pixel.

According to the name of this algorithm, we know that it is composed of two main topic. LineDiff is actually the abbreviation for Line Difference, so the main idea is to compare two consecutive lines and record the duplicated part to achieve data compression. Entropy encoding is the final phase of this algorithm, and its purpose is to produce binary code by a concept like Huffman coding. There are three major steps: LineDiff Encoding, LineDiff Compaction, and Entropy Encoding, in this algorithm and they are listed in the following.

The first step is LineDiff Encoding. In this step, line N will be encoded by some pairs in the form of (OP, L), and both OP and L are defined by comparison to line N-1 in the same position. If the data being encoded is the same as data at the same position previous row, or a duplication, we can just encode it by setting OP to DUP. Otherwise OP is set to be the color of these pixels, encoded by black, white, or a 5-bit binary number. On the other hand, L stands for length. So we set it to be the length of data that maintains this OP or we set L to END representing this OP will last to the end of this line. Figure 2.9 shows the concept of this step and Figure 2.10 is an example of LineDiff Encoding.

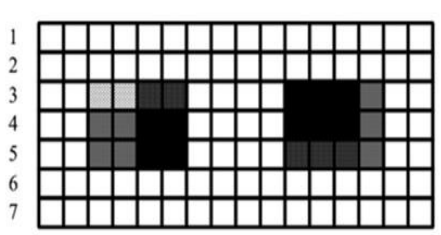
LineDiff Encoding for Scanline N:

$(OP_1, L_1)\ (OP_2, L_2)\ \ldots\ (OP_K, L_K)$

$OP = DUP$ or P, where P = white, black, or any gray value.

L = Any length value or END.

Figure 2.9 LineDiff Encoding. [1]



1. (white, END)
2. (DUP, END)
3. (DUP, 2)(7, 2)(16, 2)(DUP, 4)(black, 3)(14, 1)(DUP, END)
4. (DUP, 2)(14, 2)(black, 2)(DUP, END)
5. (DUP, 10)(16, 3)(DUP, END)
6. (DUP, 2)(white, 4)(DUP, 4)(white, 4)(DUP, END)
7. (DUP, END)

Figure 2.10 Example of LineDiff Encoding.

After LineDiff Encoding, here comes the next step called LineDiff Compaction. This step is the procedure to furthermore compress file size by omitting unnecessary data. There are three rules of LineDiff Compaction:

1) Any (OP, L) pair has L = 1, omit L.
2) First (OP, L) pair has OP = DUP, omit OP.
3) Consecutive pairs have same pixel value, combine them.

And these rules has a priority. If two or more conditions are satisfied, the order to apply these compaction rules should be 1) the first priority, then 2), and 3) be the last choice. This design is based on the next step to furthermore reduce data volume. Figure 2.11 is the previous example, we now apply these three compaction rules to them.

The last step in LineDiff Entropy is the Entropy Encoding. The author analyzed the frequency of every keyword appearance, and defined their code length by it. Table 2.1 is the analytical results in [1], and Table 2.2 is the Entropy Encoding defined by Table 2.1. One thing needed to be noticed is that this is a prefix-free encoding style, which means

that we won't be confused by prefix while decoding.

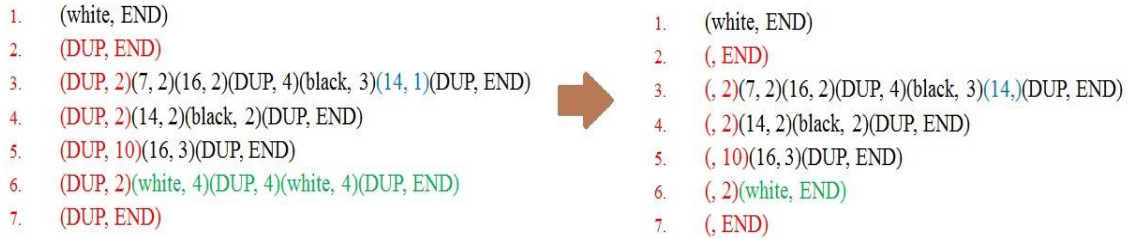


Figure 2.11 Example of LineDiff Compaction.

OP					**L**	
DUP	END	White	Black	Gray	2 – 31	32 – 1023
0.23	0.27	0.03	0.05	0.15	0.16	0.11

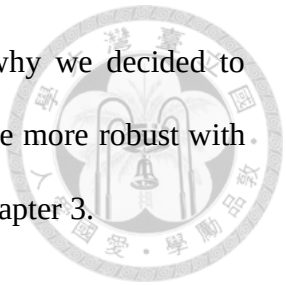
Table 2.1 Relative frequency of occurrence of OP and L. [1]

Type	**Value**	**Custom Prefix**	**# of Bits**
DUP		00	2
END		01	2
White		1000	4
Black		1001	4
Gray	1 – 30	101 + [5-bits]	8
L	2 – 31	110 + [5-bits]	8
L	32 – 1023	111 + [10-bits]	13

Table 2.2 Entropy Encoding design of LineDiff data. [1]

This LineDiff Entropy algorithm runs in linear time both encoding and decoding because it just needs one scan of all data. And its performance is extremely good both in

data compression ratio and decompression time. That is exactly why we decided to work on improving this algorithm to make MEBDW system could be more robust with it. Despite our introduction here, we'll take an even closer look in Chapter 3.



2.3 Routing Algorithms

Routing is an important step in the design of ICs. It generates wiring to interconnect pins of the same signal, on the premise that all the manufacturing design rules are obeyed. Since routing is a very complex procedure in VLSI design, we usually apply a two-stage approach of global routing followed by detailed routing to make it manageable. Global routing first partitions the routing region into tiles and decides tile-to-tile paths for all nets while attempting to optimize some given objective function (e.g., total wire length, circuit timing, and so on). Then detailed routing assigns actual tracks and vias for nets, following the guides of the paths obtained in global routing stage. In this thesis we focus on the topic of detailed routing.

There are two kinds of detailed-routing models: the grid-based and grid-less models. For grid-based routing, a routing grid is superimposed on the routing region, so the detailed router just need to find routing paths in the grids as shown in Figure 2.12, and each path among grids are called routing tracks. The space between adjacent grid lines is called wire pitch, which is defined in the technology file and is larger than or at least equal to the sum of the minimum width and spacing of wires. Note that the router has to control the searching space such that the path in the horizontal layers can only run horizontally and path in vertical layers can only go vertically for the reserved layer model. And switching from layer to layer is only allowed at the intersection of vertical and horizontal grids with vias. In this way, the wires with the minimum width following

the path in the grid would automatically satisfy the design rules. Therefore, grid-based detailed routing is much more efficient and easier for implementation [8]. So in this thesis we will not talk about grid-less detailed routing, which is basically any detailed router other than grid-based detailed router.

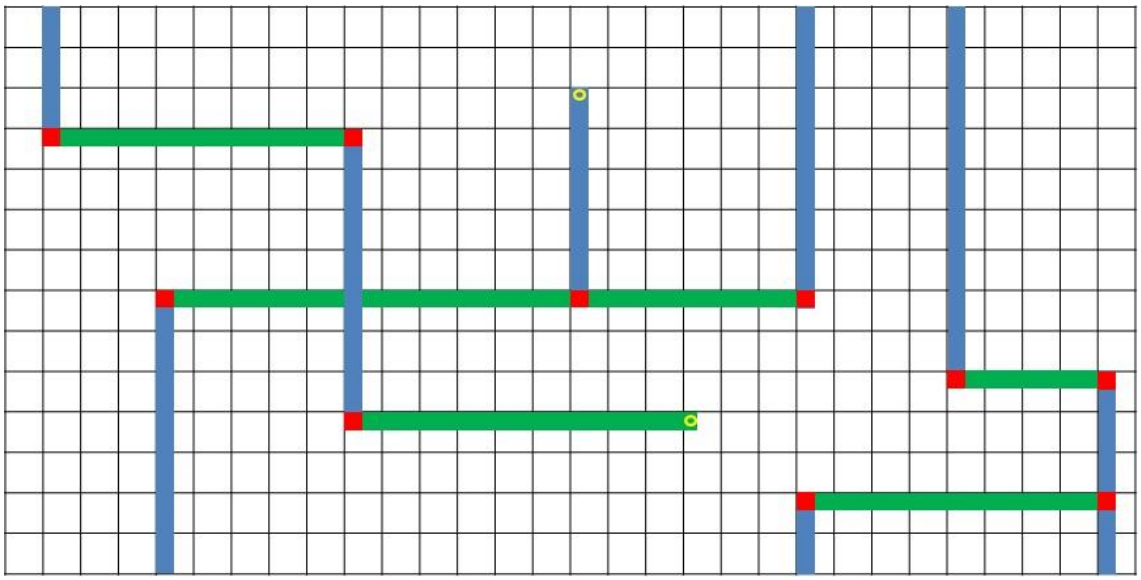


Figure 2.12 Example of grid-based routing.

With this preparation, we can now formulate our detailed routing into a path-finding problem on a grid-based map. Perhaps the most well-known algorithm for finding a path between two points is the maze-routing algorithm, or Lee's algorithm [9], which is based on the breadth-first-search (BFS) technique. After Lee, a lot of pathfinding algorithms were published and made tremendous impacts on physical design. And among all them, A* search [10] might be the most efficient and effective algorithm that widely used in routing of many different orientations [11-13]. In this section, we will give a brief introduction of both Lee's and A* search algorithm.

2.3.1 Lee's Algorithm

Lee's algorithm takes a two-phase approach of filling and retracing. Filling is the phase of work by pushing on a frontier just like wave propagation in a BFS manner. From source node S, filling stage takes procedures to fill in every grid nodes one by one by their distance of the wave-front from S, until the target node T has been arrived. Here is an example of filling stage in Figure 2.13. Figure 2.13(a) is the stage when frontier is the nodes whose distance from source is 3, and Figure 2.13(b) is frontier to be 4.

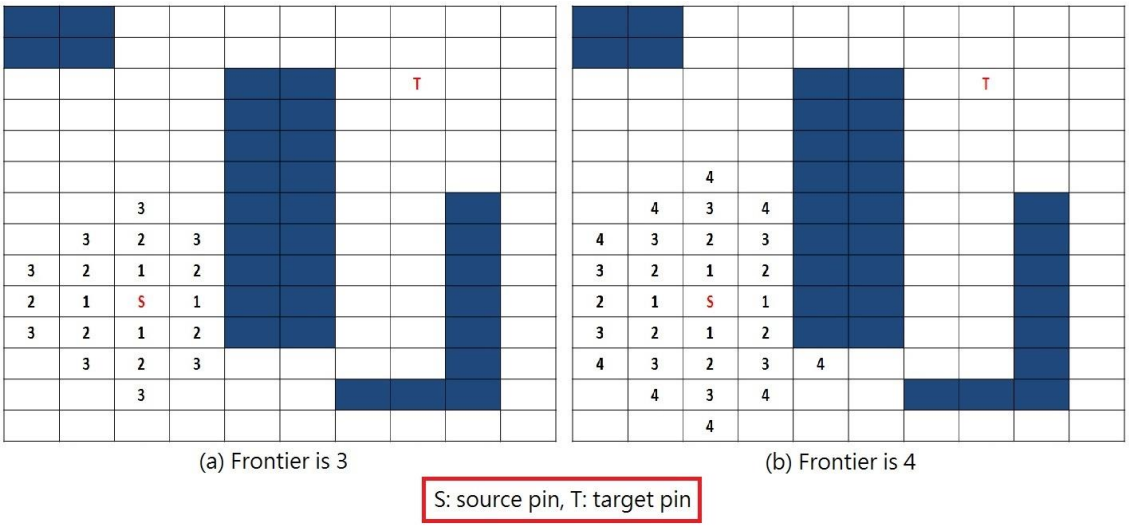


Figure 2.13 Example of filling stage in Lee's algorithm.

This filling stage goes until the target node is reached, and then a shortest path is attained by retracing from T to S in a decreasing order one node by one node while in the retracing stage. Note that there might be more than one path in the decreasing order, and each of them is a shortest path from S to T. Figure 2.14 shows an example of the retracing stage. Figure 2.14(a) is the state when the filling stage goes to reach the target node, and Figure 2.14(b) shows the retracing stage to attain the shortest path.

The best property of Lee's algorithm is that it guarantees to find a shortest path if

such path exists even with obstacles in the region. However it also suffers from the extremely-high time and space consuming shortcoming. The description above and the pseudo code in Figure 2.15 show its time and space complexity are both O(m*n), where m and n represent the number of respectively horizontal and vertical nodes, because in worst case every grid node must be labeled. And that makes it almost unfeasible to the complex networks today.

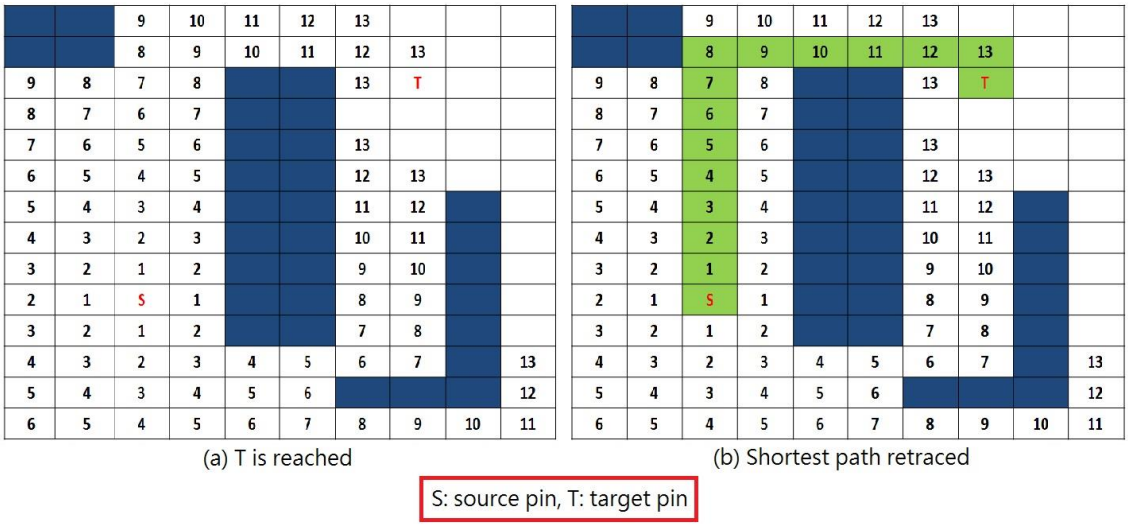


Figure 2.14 Example of retracing stage in Lee's algorithm.

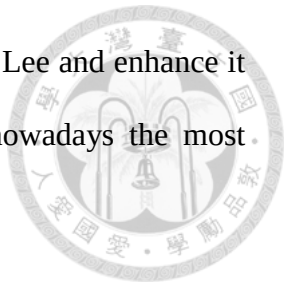
```
/*initialization*/
i := 0
The source node is labeled with i

/*filling*/
REPEAT
  - Find all nodes labeled with i
  - Label all their unlabeled neighbors with i+1
  - i := i+1
UNTIL target node reached or no more nodes to label

/*retracing*/
go to the target node
REPEAT
  - add this node into the path
  - go to its neighbor with a lower cost
UNTIL source node reached
```

Figure 2.15 Pseudo code of Lee's algorithm.

There are many grid-based shortest path algorithms inspired by Lee and enhance it in the filling complexity like Soukup [14] and Akers [15], but nowadays the most popular one must be A* search algorithm [10].



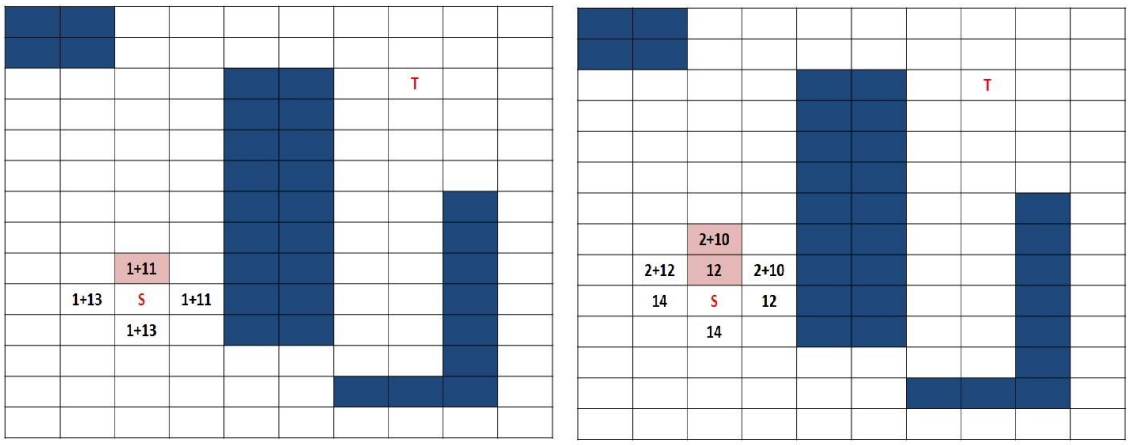
2.3.2 A* search Algorithm

As we discussed in the previous section, Lee's algorithm has a terrible time and space complexity because it basically picks the path in a blind way. One intuitive way to improve it might be to consider the nodes more likely to be in the shortest path in a prior order [8]. That is the main concept of A* search algorithm [10], and it is accomplished by adjusting the cost function in Lee's algorithm.

In Lee's algorithm, the cost function is basically the distance between current node and S. While in A* search algorithm, we define the formula f(x) = g(x) + h(x) as the cost function to evaluate the cost of each node x, where g(x) is the distance from the S to the current node x, and h(x) is the heuristic (or estimated, predicted) cost from the current node x to T. And in each round, A* search algorithm selects a node with the least cost to propagate (i.e., the least f(x)), as a result A* search is also called best-first search.

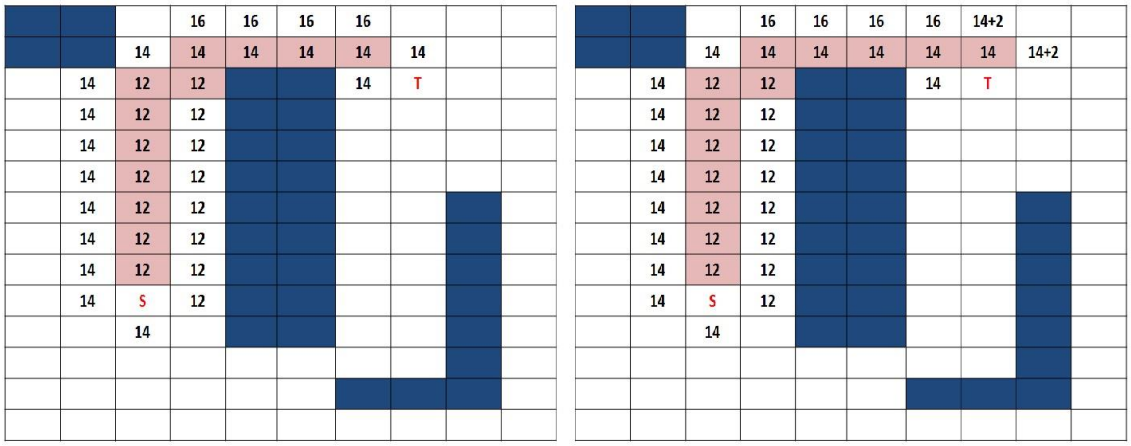
Let's take a look at an example in Figure 2.16 and Figure 2.17, here we use the Manhattan distance to estimate the h(x) part of each node. From the source node, there are only four possible nodes to propagate, and in Figure 2.16(a) we've marked all of these four neighbors by its cost function in the form of g(x) + h(x). So there are two possible choice for A* search, either one or the other has the same priority. If we choose the up one from S to propagate, there will be three more neighbors that were added to the priority queue keeping track of the path just like in Figure 2.16(b). While in Figure 2.17(a), there are also two possible nodes to be chosen. If the node which is on the top

of T is chosen, then the wave-front propagates like Figure 2.17(b), where the cost of T would be 14 + 0 = 14. So T will be popped out and the algorithm terminates. The pseudo code of A* search is showed in Figure 2.18.



(a) (b)

Figure 2.16 Example of A* search in the beginning.



(a) (b)

Figure 2.17 Example of A* search in the end.

There is another key point in the utilization of Manhattan distance as the heuristic part h(x). In [10], Hart gave a specific proof of the influence of the admissibility in this heuristic function to the whole A* search algorithm. If the heuristic part of the cost

function is admissible, which means that it never overestimate the cost of the path, then A* search algorithm always returns an optimal solution if such solution exists. Manhattan distance is the smallest possible distance between two nodes in grid-based routing, and that keeps the admissibility of the heuristic function and the optimality of A* search algorithm.

```
/*initialization*/
The source node is labeled with 0 + h(s)
add the source node into set S

/*filling*/
REPEAT
  - Find the node N with the least cost in S
  - for each N's neighbors M
    - set M.parent() to N
    - if M is not in S
      - label M with g(N) + 1 + h(M)
      - add M into S
    - else
      - update the cost of M if g(N) + 1 + h(M) is smaller than its original cost

UNTIL target node reached or S is empty

/*retracing*/
go to the target node
REPEAT
  - add this node N into the path
  - go to N.parent()
UNTIL source node reached


g(x) = the distance from source to x
h(x) = the estimated distance from x to target
```

Figure 2.18 Pseudo code of A* search.

A* search has many applications, and it also leads VLSI routing into a brand-new practical research with the appearance we see today. It perform a lot better in both time and space and that makes itself superior to other approaches. In our thesis, we not only modify this algorithm to meet our requirements, but also implement it to be the control group as a basis for comparison.

Chapter 3 Data Compression Ratio-aware Detailed Routing

This chapter is the main theme of this thesis. In the following sections, we will introduce a detailed router which is capable of improving data compression ratio while LineDiff Entropy data compression algorithm is used. Our routing algorithm is also based on A* search, but three more strategies are proposed up onto it. The first strategy is to make the wires on-grid after pixelization, but this would cause a decrease in routability. So we propose the second strategy to enhance routability and to supply a reference for the third strategy, which is to route the patterns more expected to enhance total data compression ratio. All the details will be illustrated in the following sections right after the introduction of the routing specification.

3.1 Routing Specifications

International Technology Roadmap for Semiconductors (ITRS) [13] shows that in 14-nm technology node, half wire pitch is expected to be 40nm. Wire pitch is defined as in Figure 3.1, so in the routing spec., we can make our wires to be 40 nm, and a routing track is therefore 80 nm to avoid design rule violation.

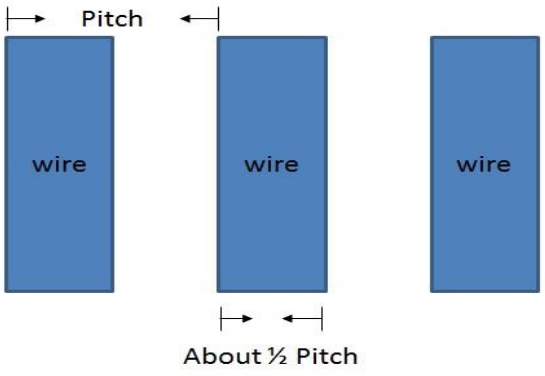


Figure 3.1 Narration of wire pitch.

And just like what we've talked about in section 1.1, our pixel size is set to be 7nm by 7nm. Each pixel contains a 5-bit data, which gives a precision of 32-level grayscale. Total routing spec. is shown in Table 3.1.

14/16-nm technology node	
Wire width	40 nm
Track width	80 nm
Pixel size	7nm * 7nm
Pixel depth	5 bits
# of layers	3
Expected CR	450

Table 3.1 Routing specification.

MEBDW systems are relatively costly and slow compared to traditional process, so we usually use them in the layers with the thinnest wires. In our research, we assume the number of this kind of layers to be three, including two horizontal-orientation layers and one vertical layer in a staggered order. Expected Compression Ratio (CR) is also attained from section 1.1, which is the ratio of required optical fibers to the actual number. This spec. is also implemented to be the control group with merely A* search algorithm and the 450 times of compression ratio is the ultimate goal we want to achieve.

3.2 1st strategy: on-grid wires

The first thing we do for this data compression ratio-aware router is to make the wires we route can be exactly on the grid lines after pixelization. Figure 3.2 is a small example of the benefit we can get with this strategy. In the figure, layout on the right is

the one with some special process to make the wires lies just on the gridlines, while the left one isn't. And the binary digits beneath them are the encoded results with LineDiff Entropy algorithm.

We can see that in a layout, a piece of wire is composed of two parts. The interior pixels are all set to be 31 (Black) because all of the area in these pixels is occupied by the wire, while the pixels surrounding the wire are all set to be some number between 1 and 30 because their area are just partially occupied. And we also notice that these surrounding pixels does burden CR a lot because of their fragmented information. If we can make our nets lie on grid lines as much as possible like in the example in Figure 3.2, we can get the results of a 1.87 times of improvement in CR.

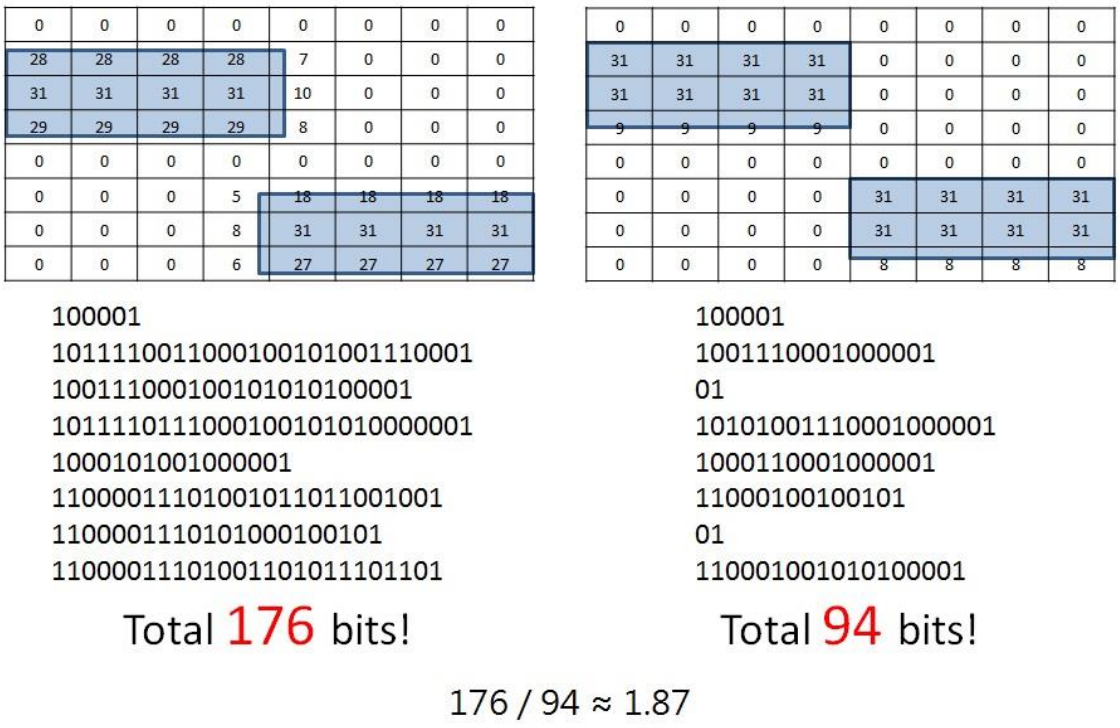


Figure 3.2 Example of nets on grid lines.

To put this idea into practice, we modify the routing specification we've described in the previous section. In section 3.1, we've talked about that ITRS expected the wire pitch (hence the track width) to be at least 80 nm and a pixel is of 7-nm in each side.

With this spec., it is difficult to make wires lie on the exact grid lines. So what we did is to adjust the routing track to be 84 nm which is divisible by 7 and make the wires on the top-left corner of each track. With this little technique, we can now make sure that our metal wires will always stick on the grid lines for up, left, and right sides.

Figure 3.3 shows the concept of the first strategy. Figure 3.3(a) is the layout with original track width, while (b) is the layout after adjustment. We can see that the track width in (b) is wider than (a), so the requirements in ITRS roadmap are automatically achieved at the cost of fewer routing tracks. After the process of pixelization, (a) becomes (c), and (b) becomes (d). It is obvious that the edges of all nets in (d) are right on gird lines, therefore CR is much better than (c).

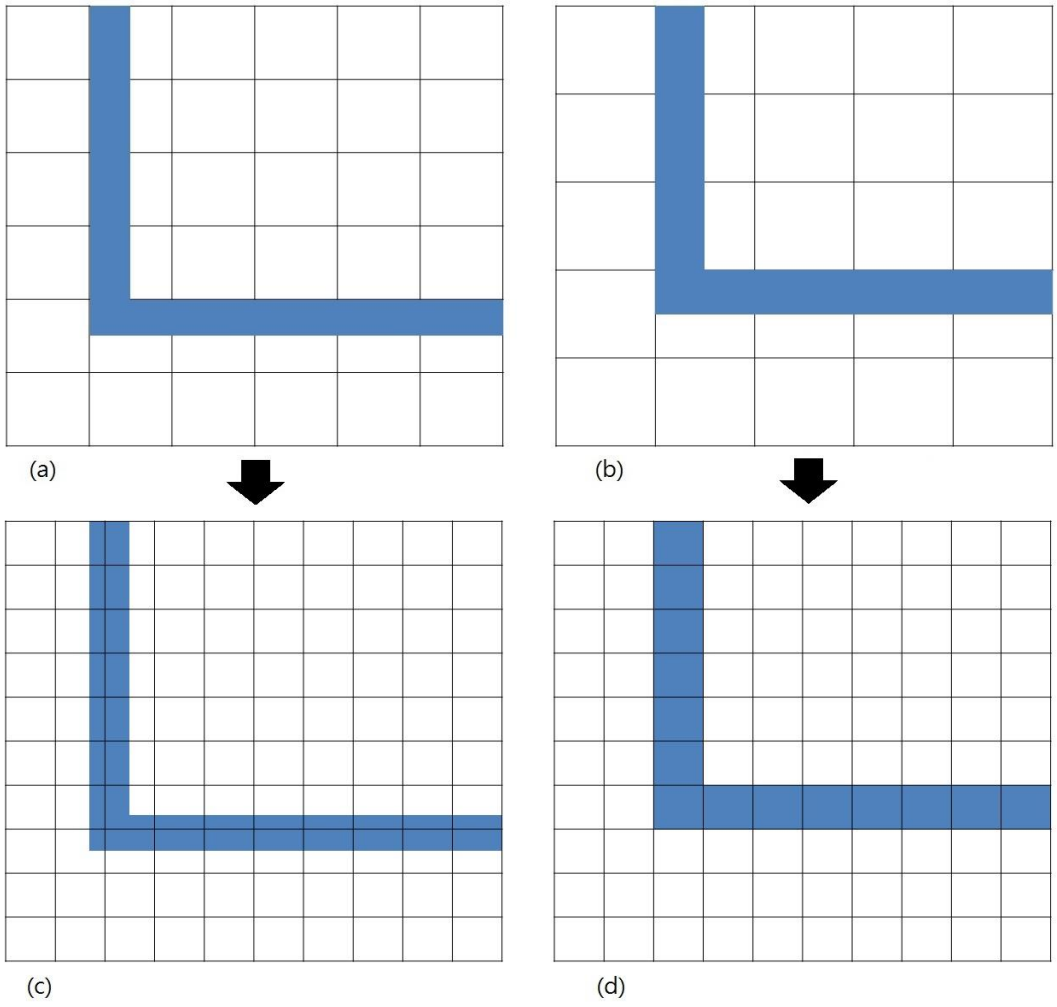


Figure 3.3 Concept of the first strategy.

We adjust the size of each routing grids to make the nets on-grid so that the compression ratio could be improved, but the number of routing tracks is therefore reduced. That might cause a severe problem in routability, so we propose the second strategy to ease this problem.

3.3 2nd strategy: simple route

The second strategy is a process called simple route. We've already known that A* search is a shortest-path finding algorithm. And as the description in our routing spec., we route in three layers containing two layers of horizontal wires and a layer of vertical in between, just like shown in Figure 3.4. So it is reasonable to speculate that the nets we route with A* search would congest in Layer 0 and Layer 1 since all pins are defined in Layer 0. This and the problem we've talked about in section 3.2 cause a drop in routability.

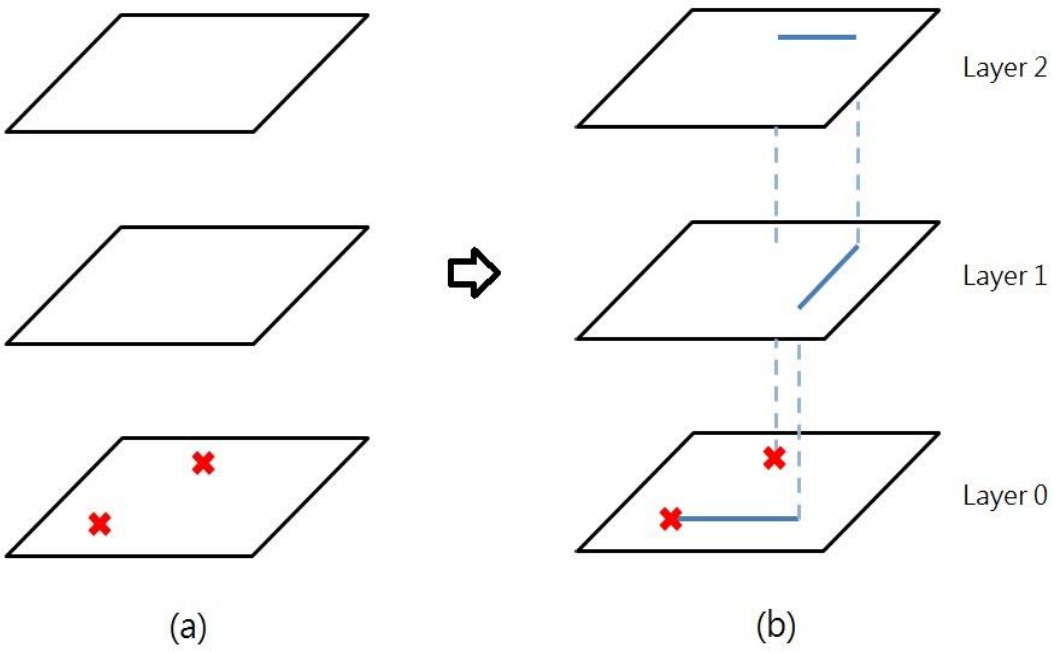


Figure 3.4 Illustration of our 3-layer routing scheme,

X denote pins to be connected.

To alleviate this routability problem, we've designed a simple route process to enhance the utilization of Layer 2. Whenever we are going to route a new wire, we check whether we can go directly by Layer 1 and Layer 2 just like Figure 3.5. In the figure we illustrated two different types of simple route, each of them routes in an opposite order of vertical wires and horizontal wires. Both of them are used in the proposed routing algorithm to enhance routability and Layer 2 area usage. Note that A* search algorithm is used if we cannot apply either of these two simple route processes.

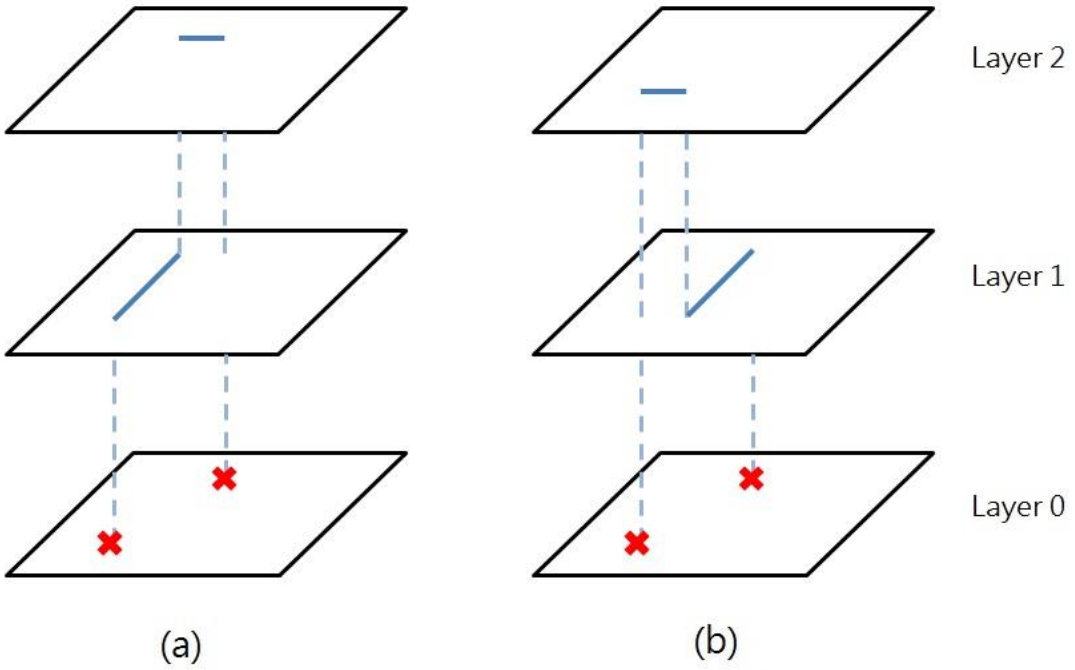


Figure 3.5 Examples of simple route.

With this technique, proposed router did surpass the control groups in routability. We'll talk about the experiment results in the later chapters. Another benefit of this simple route process is that our algorithm routes some layout pattern that is long and steady first, and these patterns themselves can be a great reference when we want to route some patterns we really want in the next strategy.

3.4 3rd strategy: desired patterns

In addition to the two previous strategies, the most important part must be how to route the patterns with better compression ratio. In the description in 2.2.2, we've introduced LineDiff Entropy compression algorithm in detail. And in 2.3.2, we know that A* search algorithm takes advantage of cost functions to find a path between the source pin and the target pin. So in this section, the main purpose is to first find out what patterns perform well with LineDiff Entropy, and then we'll make an adjustment in the cost functions to make our router produce these patterns more likely.

Because LineDiff Entropy makes use of the duplication of layouts to compress data volume, it is trivial that layout with more repetitive patterns has a better CR. Here we'll talk about layout pattern performance with horizontal layer (Layer 0, Layer 2) and vertical layer (Layer 1) separately.

In Layer 0 and Layer 2, only horizontal wires are allowed. Our strategy is to make sure that a routing track is more likely to be selected if it had been used before. This idea comes from Figure 3.6(a) and (b). We can see that there are both three wires in these two figures, but (b) is more likely to have a better CR because all wires are in the same track. So LineDiff Entropy would take a little effort to encode this one track and then just duplicate all the blank tracks, which might results in a great CR.

And there is a similar situation in Layer 1 which only vertical-oriented wires are allowed. In Figure 3.6(c) and (d) we can see that there are also three wires in both figures but (d) should perform better in CR. Because patterns in (c) is not well-aligned as in (d), LineDiff Entropy cannot encode them only by their duplication. So we can conclude that in the layer of vertical wires, our strategy is also to make the best out of the duplication.

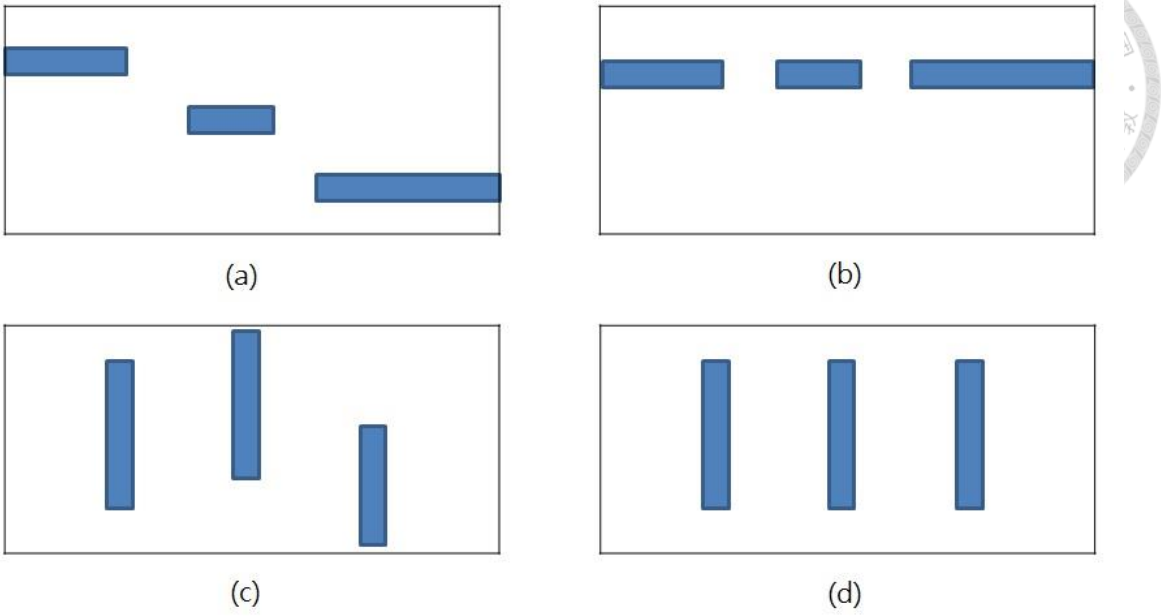


Figure 3.6 Expected routing patterns.

Because LineDiff Entropy needs input file which is 1024-pixel wide and a pixel is 7 nm each side, the first specific procedure to do the above is to split the whole routing map into several stripes which is 7168 nm wide. And then give the routing grids that is not forming our desired pattern a punishment in the cost. In this case, A* search would more likely to use the desired grid because it always picks the node with least cost to propagate.

Figure 3.7 shows an example of Layer 1 (vertical-wire allowed). In this figure, the whole layout is split into 7168-nm-wide stripes. The blue obstacle is a wire that had been routed already and the yellow one is the one to be routed. There are two possible directions A and B for the wire to propagate because it can go only vertically. And in this case, the cost of B would be raised by a punishment because if the router route that path, LineDiff Entropy should spend more resource to compress.

So what we actually do is that we keep trace of wires routed in each stripe horizontally because of the raster scan order of LineDiff Entropy. In A* search, we can

say that we add the candidate nodes of the next propagation into a priority queue to pick the one with least cost. And in our algorithm, we check whether the row where the node lies in has been used beforehand or not. If not, we add a punishment to the cost of it. For example, in Figure 3.7, we've record the blue wire. So when we are routing the yellow wire, we check the record to know that the row of node A has been used, while the row of node B hasn't. So it is more expected to use A than B for LineDiff Entropy performance. We raised the cost of node B by a punishment value to make the router go for A more likely.

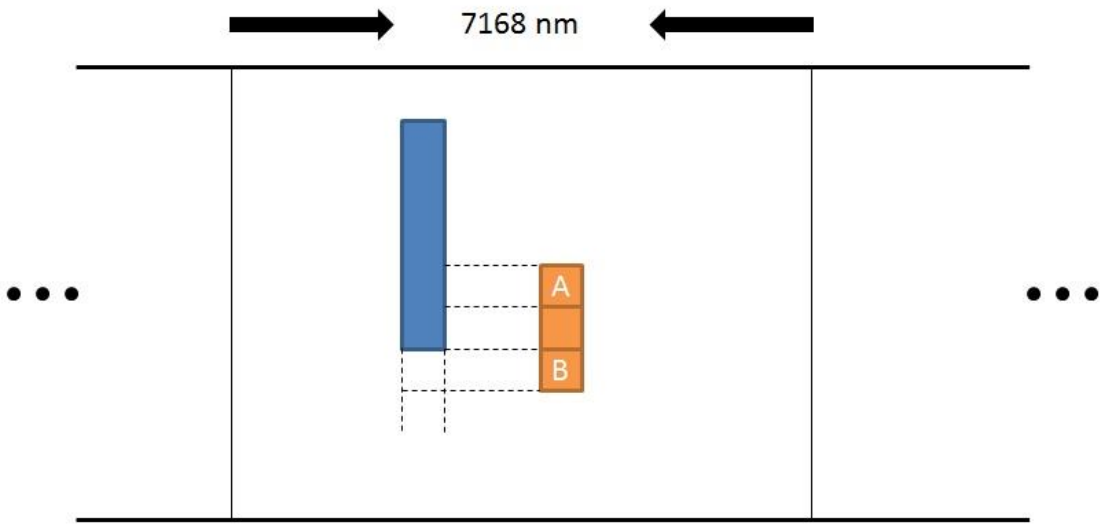


Figure 3.7 Example of the third strategy.

Another important issue we want to talk about is that in section 3.2 we have mentioned our routing track is set to be 84 nm. So if we want to split the layout into stripes with 7168 nm, each stripe would contain about 85.33 tracks, which is not processable.

We dealt with this problem by the expression in Figure 3.8. We try to solve it by the way as simple as possible, so all stripes are divided into groups of three. The first

and the third stripes contain 85 tracks, and the second stripe contains 86. So every three stripes contains 256 tracks, which is exactly 21504 nm for three stripes.

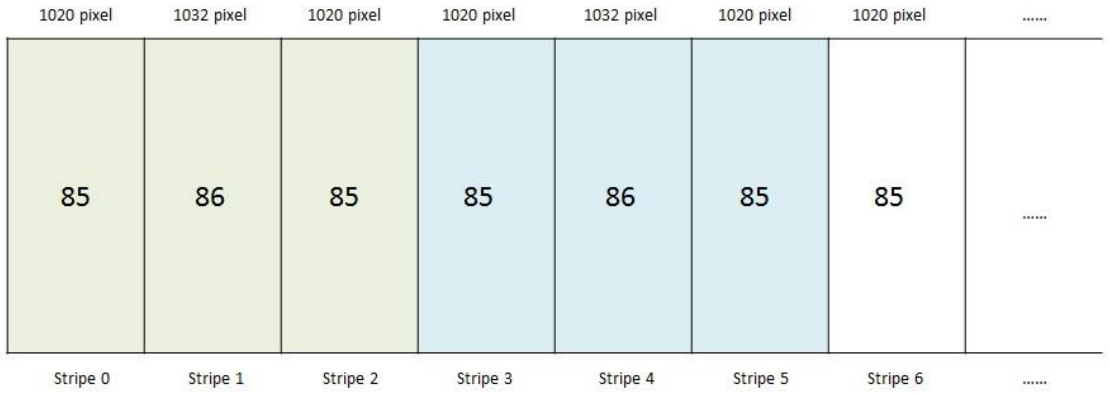


Figure 3.8 Stripe splitting.

In the end of this chapter, Figure 3.9 shows the whole flow chart of the proposed algorithm except for the track width adjustment. In the figure we know that we first use simple route in both directions, and the modified A* search carries out if both of them failed. The key procedure of this modified A* search is the cost calculation because we use the information of not only distance but the applicability of LineDiff Entropy to determine the cost, and that makes our proposed detailed routing meaningful and effective. We'll give some experiment results to support this statement in the next chapter.

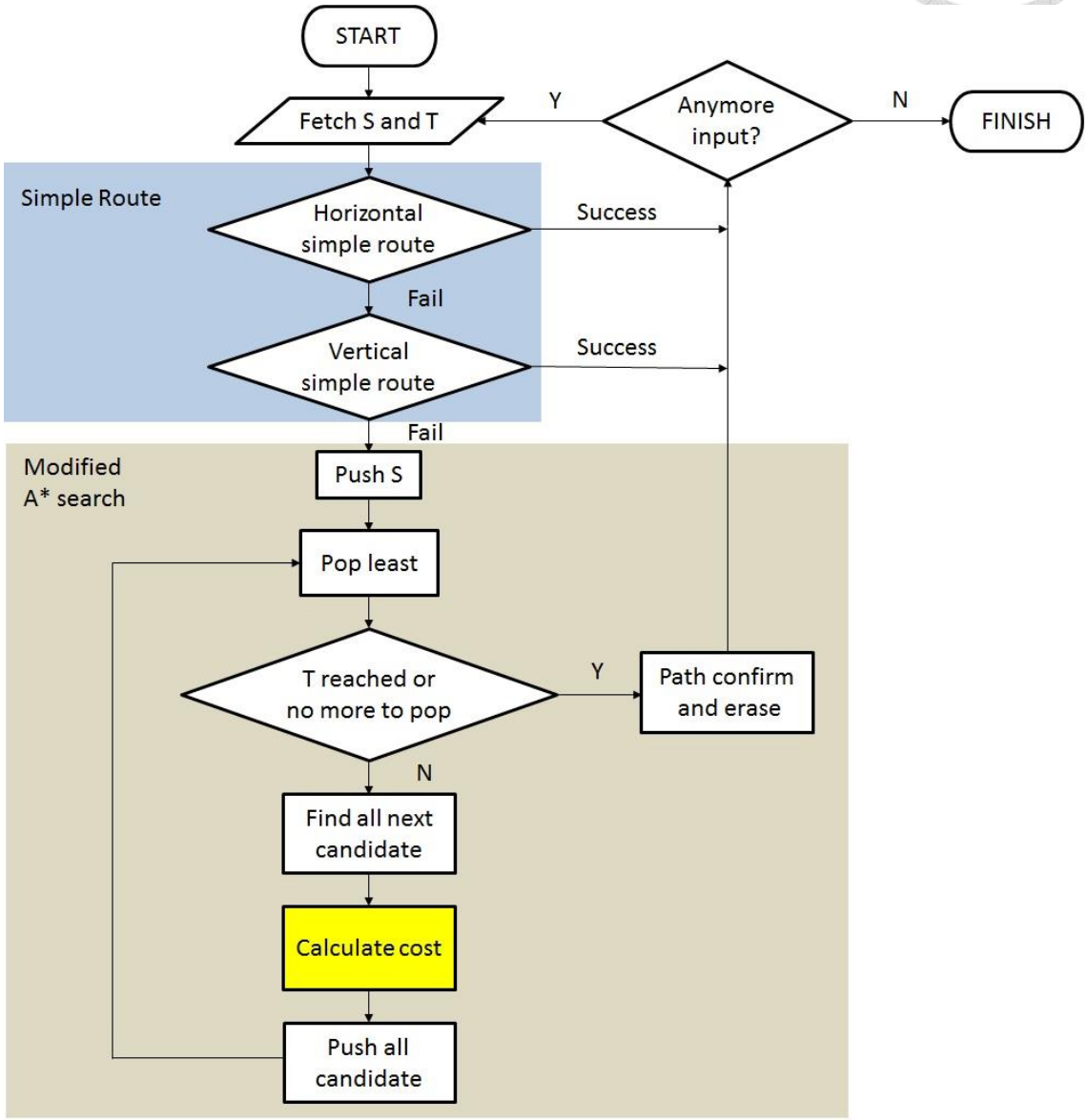


Figure 3.9 Total flow chart of the proposed algorithm.

Chapter 4 Results of Experiments

In this chapter, the results of experiments we collected will be illustrated. And all of them are conducted on workstation running Linux with Intel(R) Xeon(R) CPU E5-2643 v2 3.50 GHz system with 70 GB RAM. Both proposed algorithm and the control group, along with LineDiff Entropy algorithm, were implemented in C++ programming language compiled by g++ (GCC) v.4.8.2 released on 20140120.

Table 4.1 is adopted from Table 3.1, and it shows the specification of our implementation. Basically our proposed router and the control group are both designed for 14/16-nm technology node just like we introduced in section 3.1. The only difference between them is the strategies we've expressed in the previous chapter. The first strategy needs the routing track to be 84 nm in the width, resulting in a drop in routability. So a procedure which checks whether the source pin is able to connect the target pin directly, simple route, is taken place. But the most important modification is the calculation of the cost in A* search. Proposed router takes advantage of both distance and the performance of the patterns in LineDiff Entropy algorithm. While the control group in our implementation is a detailed router which contains only the original A* search algorithm to find the shortest path. And all other specifications in both designs are just the basic requirements in the technology roadmap. Expected compression ratio is 450, estimated in section 1.1 and it's the ultimate goal we are trying to reach out.

The input cases were generated at random by their coordinates, and the original point was set on the top-left corner. Figure 4.1 shows a concept of this input file generation. There are three pairs of source and target pins in this figure, and our detailed router should find paths to connect them separately without any intersection. Another

constraint set in generating input is that each source and target node cannot be in the same grid in neither proposed router nor control group. Note that this figure does not show the exact input file format, the actual format is just pairs and pairs of coordinates of pins.

Groups	Proposed	A* search
Track width	84 nm	80 nm
Simple route	Yes	No
Cost evaluation	Distance and LDE(*) preference	Distance only
Wire width	40 nm	
Pixel size	7nm * 7nm	
Pixel depth	5 bits, 32 gray-level scales	
Number of layers	3 layers	
Expected CR	450	

Table 4.1 Spec. of the implementation,

(*)LDE: LineDiff Entropy algorithm.

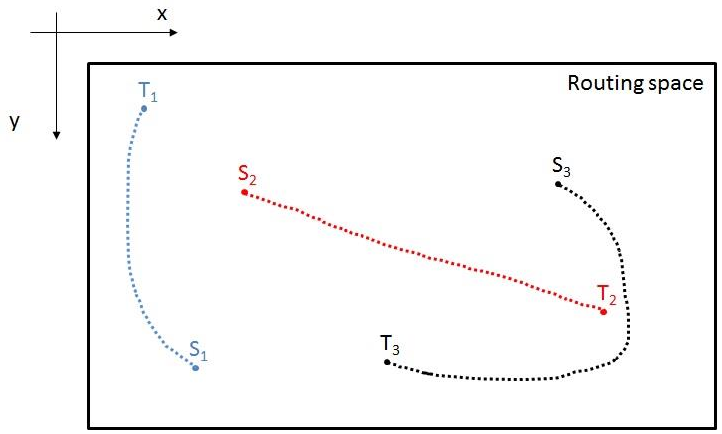
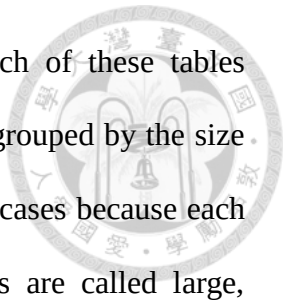


Figure 4.1 Concept of input generation.

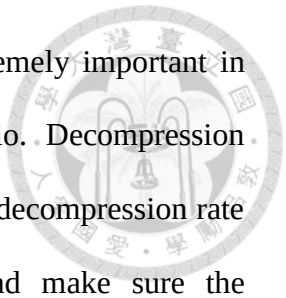


The following tables are the results of our experiments. Each of these tables contains four sets of input cases generated at random, and they are grouped by the size of the input circuit. We named the input cases in Table 4.2 the huge cases because each of them contains 10,000 wires to be routed. And the other cases are called large, medium, small, and tiny cases for Table 4.3, Table 4.4, Table 4.5, and Table 4.6 respectively, because of the number of wires needing routed is 8000, 5000, 3000, and 1000. After these complicated information, Table 4.7 shows the organized result of all.

In these tables, the basic information of each cases is listed on the top. The identifiers of each input cases were given just for convenience. In each cases, all the results can be viewed by two parts. The information on the left is from the control group which was implemented using A* search algorithm only, while the one on the right-hand side is our proposed detailed router. There are four topics we want to discuss, including routability, total wire length, and the two most-important issues in MEBDW systems, decompression rate and data compression ratio in LineDiff Entropy.

Routability is the primary concern for every router, and it's defined by the success rate while finding the actual path for each wire. In the proposed detailed router, we designed a simple route process in order to deal with the severe problem in routability caused by making wires on-grid. And the results show that the proposed algorithm does route more wires than pure A* search. Actually the proposed detailed router has routability more than 95% in every single test case and 97.59% on average.

But everything has its price. To reach such a high routability and the benefits in other aspects, the router usually needs to take longer detours. So in Table 4.7 we can see that the total wire length in the proposed algorithm is 23.43% more than the control group. It's actually not negligible, but compared to the contribution of this work, we should try to do something more in the future to fix this.



The two issues we want to discuss next, which are also extremely important in MEBDW systems, are decompression rate and compression ratio. Decompression hardware should be directly attached to the electron beam writer, so decompression rate being as fast as possible can save the hardware resources and make sure the decompressed data sent to the writers in real time. In our experiment results, we can see that the decompression time in the proposed router and the control group are almost the same. So we can conclude that our design did not affect the decompression rate.

As for the compression ratio, just like we estimated, has a great improvement. The compression ratio (CR) here is defined as the original data size divided by the compressed data size. The results show that CR in proposed router is improved by at most 1.67 times than the router with only A* search. And in Table 4.7 we can also know that our design has a roughly 1.42 times on average better than the control group in CR.

But aside from the multiples we've mentioned above, the important part is the goal of our design. In the previous section we know that the desired CR is about 450 in manufacturing. The results also show that in most of the input case, proposed router reach this target, while the control group doesn't.

These two important issues are also plotted in Figure 4.2 and Figure 4.3. Note that in these figures, the numbers had been taken average in each size of input. We can see in Figure 4.2 that the decompression time is nearly the same in both designs. And Figure 4.3 shows CR in both designs, with the black line in the middle shows the ultimate goal, 450 times, of compression ratio in our research.

Input case ID.	Huge_#1		Huge_#2	
# of wires	10000		10000	
Area (nm^2)	97440 * 97440		97440 * 97440	
Group	A* search	Proposed	A* search	Proposed
Routability	84.03%	93.40%	91.55%	97.48%
Total wire length (nm)	29614800	37031228	14799960	18289352
Decompression time (s)	38.51	37	37.02	37.72
Original data size (Byte)	1796598720	1796598720	1796598720	1796598720
Compressed data size (Byte)	6795886	4066245	5513903	3660736
Compression ratio	264.3656353	441.8323834	325.830672	490.7752758
CR Improvement	1.671292802		1.506227983	

Input case ID.	Huge_#3		Huge_#4	
# of wires	10000		10000	
Area (nm^2)	97440 * 97440		97440 * 97440	
Group	A* search	Proposed	A* search	Proposed
Routability	91.74%	97.73%	88.95%	96.05%
Total wire length (nm)	14773440	18223484	19912040	24538916
Decompression time (s)	37.82	37.24	34.74	35.39
Original data size (Byte)	1796598720	1796598720	1796598720	1796598720
Compressed data size (Byte)	5539075	3655172	5962215	3807019
Compression ratio	324.3499537	491.522347	301.3307504	471.9174556
CR Improvement	1.515407483		1.566111175	

Table 4.2 Results of experiments for huge cases.

Input case ID.	Large_#1		Large_#2	
# of wires	8000		8000	
Area (nm^2)	87360 * 87360		87360 * 87360	
Group	A* search	Proposed	A* search	Proposed
Routability	85.81%	94.34%	92.81%	97.89%
Total wire length (nm)	21290680	26586708	10617960	12997012
Decompression time (s)	30.83	30.84	31.82	32.69
Original data size (Byte)	1495690560	1495690560	1495690560	1495690560
Compressed data size (Byte)	5277262	3252139	4338296	2947402
Compression ratio	283.4216986	459.9097886	344.7645251	507.4606586
CR Improvement	1.622704934		1.471905088	

Input case ID.	Large_#3		Large_#4	
# of wires	8000		8000	
Area (nm^2)	87360 * 87360		87360 * 87360	
Group	A* search	Proposed	A* search	Proposed
Routability	92.28%	98.01%	89.76%	96.65%
Total wire length (nm)	10520240	12951368	14125160	17563512
Decompression time (s)	31.75	31.41	29.49	30.12
Original data size (Byte)	1495690560	1495690560	1495690560	1495690560
Compressed data size (Byte)	4333774	2940447	4708042	3062791
Compression ratio	345.1242635	508.6609485	317.6884488	488.3423518
CR Improvement	1.473848704		1.537173774	

Table 4.3 Results of experiments for large cases.

Input case ID.	Medium_#1		Medium_#2	
# of wires	5000		5000	
Area (nm2)	67200 * 67200		67200 * 67200	
Group	A* search	Proposed	A* search	Proposed
Routability	88.10%	95.74%	93.60%	98.30%
Total wire length (nm)	10160440	12623972	4951760	5968560
Decompression time (s)	18.35	19.1	17.45	17.45
Original data size (Byte)	885024000	885024000	885024000	885024000
Compressed data size (Byte)	2980874	1892347	2414204	1714069
Compression ratio	296.9008418	467.6858948	366.5903958	516.3292726
CR Improvement	1.575225897		1.40846372	

Input case ID.	Medium_#3		Medium_#4	
# of wires	5000		5000	
Area (nm2)	67200 * 67200		67200 * 67200	
Group	A* search	Proposed	A* search	Proposed
Routability	93.46%	98.64%	91.40%	97.38%
Total wire length (nm)	4921960	5973624	6738560	8327120
Decompression time (s)	17.98	17.48	18.47	18.08
Original data size (Byte)	885024000	885024000	885024000	885024000
Compressed data size (Byte)	2426859	1721081	2661106	1798012
Compression ratio	364.6787885	514.2256524	332.5775072	492.2236337
CR Improvement	1.410078317		1.48002683	

Table 4.4 Results of experiments for medium cases.

Input case ID.	Small_#1		Small_#2	
# of wires	3000		3000	
Area (nm^2)	52080 * 52080		52080 * 52080	
Group	A* search	Proposed	A* search	Proposed
Routability	91.07%	96.90%	95.20%	99.07%
Total wire length (nm)	4700520	5883716	2269280	2628396
Decompression time (s)	10.98	11.38	10.85	10.55
Original data size (Byte)	548714880	548714880	548714880	548714880
Compressed data size (Byte)	1703269	1110990	1337456	1010446
Compression ratio	322.1539757	493.8972268	410.2676125	543.0422605
CR Improvement	1.533109209		1.323629368	

Input case ID.	Small_#3		Small_#4	
# of wires	3000		3000	
Area (nm^2)	52080 * 52080		52080 * 52080	
Group	A* search	Proposed	A* search	Proposed
Routability	94.30%	99.20%	93.30%	98.67%
Total wire length (nm)	2234960	2648696	3063680	3672492
Decompression time (s)	10.8	10.45	11.34	11.29
Original data size (Byte)	548714880	548714880	548714880	548714880
Compressed data size (Byte)	1332523	1008515	1489779	1058049
Compression ratio	411.7864232	544.0820216	368.3196501	518.6100833
CR Improvement	1.321272366		1.408043484	

Table 4.5 Results of experiments for small cases.

Input case ID.	Tiny_#1		Tiny_#2	
# of wires	1000		1000	
Area (nm^2)	30240 * 30240		30240 * 30240	
Group	A* search	Proposed	A* search	Proposed
Routability	95.00%	97.80%	97.50%	99.30%
Total wire length (nm)	881440	1072176	421040	427356
Decompression time (s)	3.97	4.14	3.89	3.8
Original data size (Byte)	199130400	199130400	199130400	199130400
Compressed data size (Byte)	497082	361671	383569	316350
Compression ratio	400.598694	550.5843709	519.1514434	629.4623044
CR Improvement	1.374403809		1.212483009	

Input case ID.	Tiny_#3		Tiny_#4	
# of wires	1000		1000	
Area (nm^2)	30240 * 30240		30240 * 30240	
Group	A* search	Proposed	A* search	Proposed
Routability	98.10%	99.60%	97.20%	99.60%
Total wire length (nm)	437640	439072	572360	625980
Decompression time (s)	4.13	3.79	3.85	4.01
Original data size (Byte)	199130400	199130400	199130400	199130400
Compressed data size (Byte)	389990	318149	423360	335303
Compression ratio	510.6038616	625.9029574	470.3571429	593.8819515
CR Improvement	1.225809291		1.262619183	

Table 4.6 Results of experiments for tiny cases.

Group	A* search	Proposed
Average routability	92.26%	97.59%
Total wire length (nm)	177007920	218472740
Total wire length exceeded	23.43%	
Total decompression time (s)	404.04	403.93
Difference of decompression time	-0.03%	
Average compression ratio	364.0431	517.5174
Average CR improvement	1.421583	

Table 4.7 Overall results of experiments.

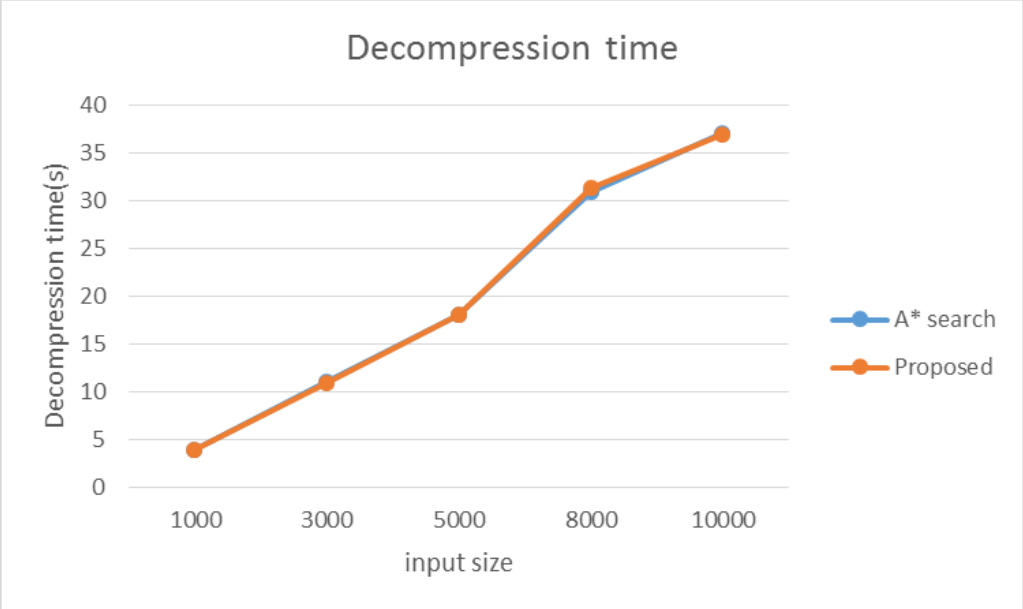


Figure 4.2 Decompression time comparison

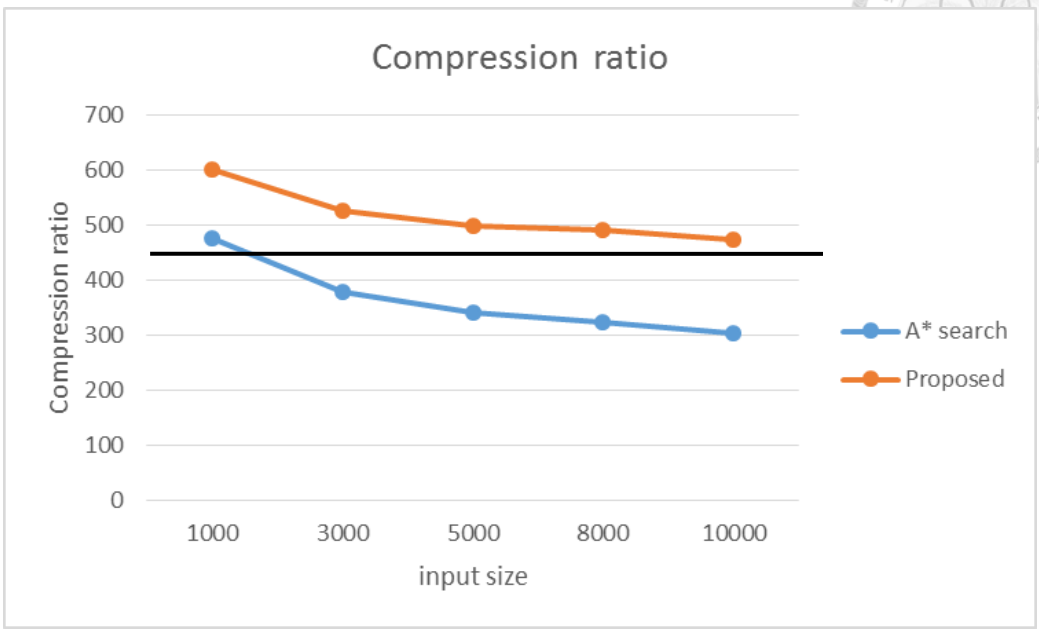
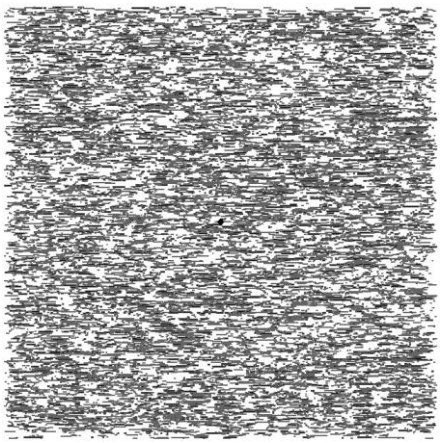


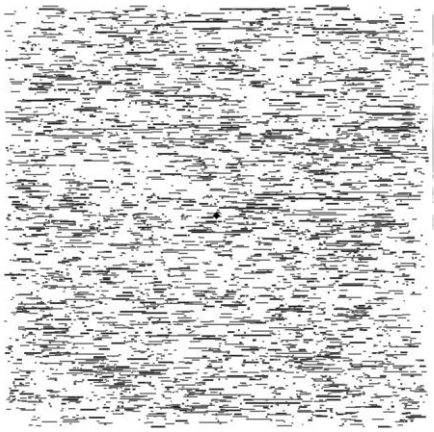
Figure 4.3 Compression ratio comparison,

the black line indicates the goal of the research 450

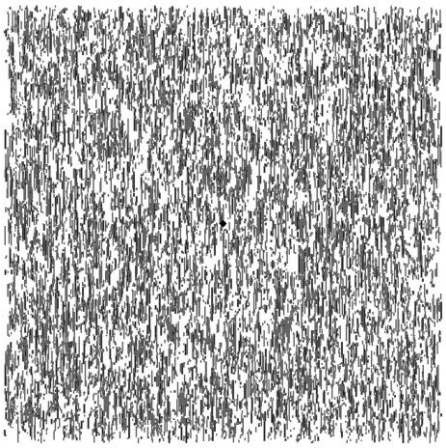
There are two routing results shown in Figure 4.4 and Figure 4.5. In both figures, the layouts on the left is the result of the control group, while the ones on the right is the result of our proposed router. And all three layers are listed together. We can see that when we're using only A* search, layer_2 is rarely used, while proposed router has a better area utilization. Figure 4.4 is the result of case Huge_#1 and Figure 4.5 shows the result of case Medium_#3.



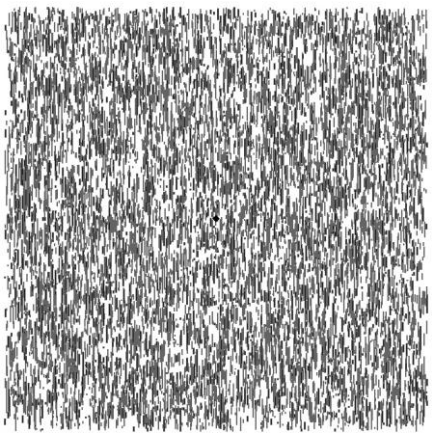
Layer_0



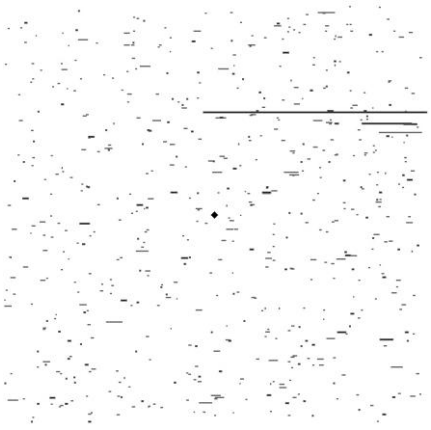
Layer_0



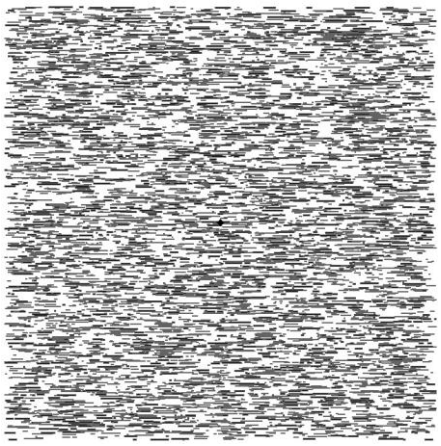
Layer_1



Layer_1



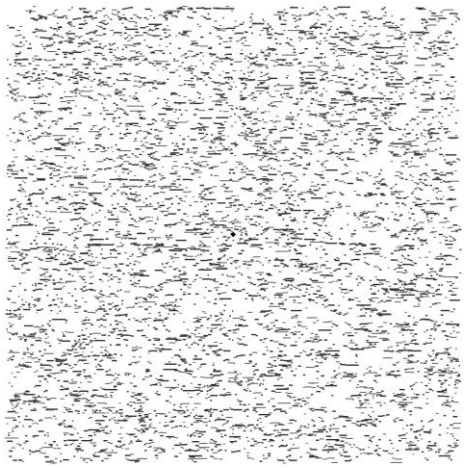
Layer_2



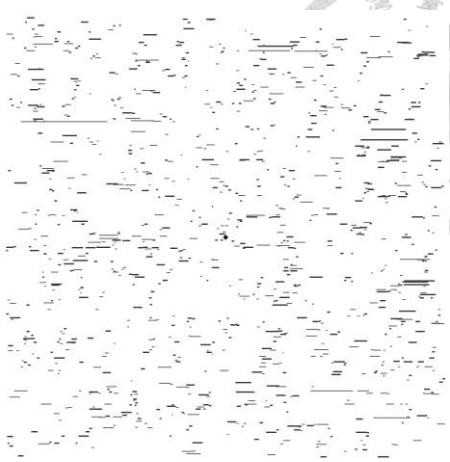
Layer_2

(a) A* search (b) Proposed

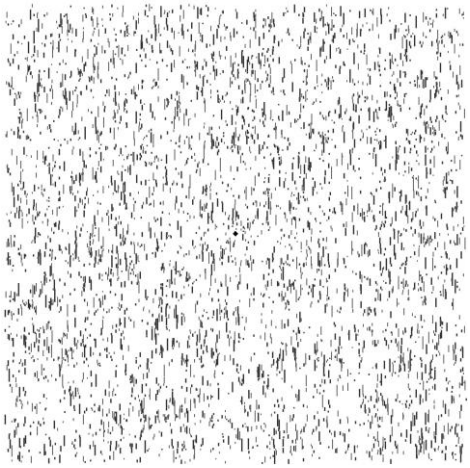
Figure 4.4 Routing results for Huge_#1.



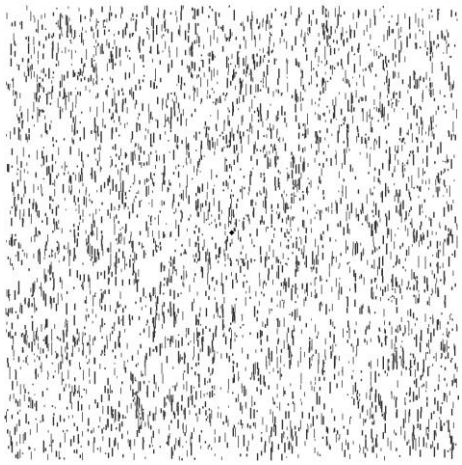
Layer_0



Layer_0

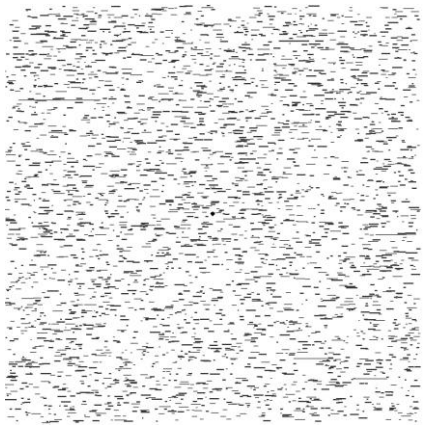


Layer_1



Layer_1

Layer_2



Layer_2

(a) A* search

(b) Proposed

Figure 4.5 Routing results for Medium_#3.

Chapter 5 Conclusion and Future Work

In this thesis we have proposed a detailed router that supports the data compression algorithm, LineDiff Entropy algorithm, for MEBDW systems. In the beginning we have introduced such system including the significance, the expectation, and the problem it confronts. And then we did some calculation to estimate the lack in throughput needs a compression ratio larger than 450.

After the complete expression of the algorithms, the results of experiments were listed. And they've shown that for LineDiff Entropy algorithm, proposed detailed router performs much better than the router without our modification in compression ratio while maintaining decompression rate. And the estimated goal 450 is reached after applying our algorithm.

And there is another important contribution in our research. We've mentioned that there is no such research in solving throughput problem in MEBDW system from the aspect of circuit physical design. Our research proved that this is a field worth investment.

But there also exists something more to improve. For routing part, routability should be further promoted to around 100% to meet the requirements in manufacturing these days. And this could be done if we introduce a structure which is able to dynamically select, defuse and reroute. Such structure could automatically selects wires causing routing congestion or making LineDiff Entropy performs poorly. And then these wires might be canceled in order to route other wires first, to further improve routability or CR.

As for industrial manufacturability, there is actually an important phase we ignored in this thesis called Electron-beam Proximity Correction (EPC). This phase is necessary

because the patterns might have a distortion after written by E-beam emitter.

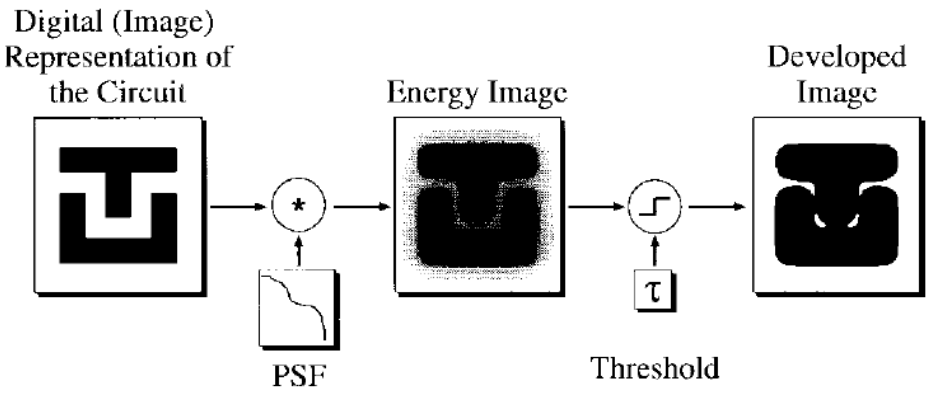


Figure 5.1 Latent image simulation procedures. [17]

Figure 5.1 shows the simulation of a pattern distortion. In this flow, the original layout is represented by some standard polygons, and then they're taken convolution with a Point Spread Function (PSF) which can be derived from Gaussian function [18] to get the energy map. And finally after filtered by a threshold, we get the image on the right in Figure 5.1, which is the actual pattern that will be written. We can see such a variation that even causes circuit short.

This is why the EPC process is needed. But we can also comprehend that after EPC, layout patterns could have skewed shapes which cannot be compressed easily by any data compression algorithm.

Another work might be worth researching is a brand new data compression algorithm which can cooperate with our proposed router or with little adjustment. Because the basic idea of proposed router is to take advantage of the repetition of patterns, there should be some more ways to further improve compression performance with it.

REFERENCE

[1] Chin-Khai Tang, Ming-Shing Su, and Yi-Chang Lu, “LineDiff Entropy: Lossless Layout Data Compression Scheme for Maskless Lithography Systems,” *IEEE Signal Processing Letters*, Vol. 20, No. 7, July 2013, pp. 645-648.

[2] P. Petric, C. Bevis, A. Brodie, A. Carroll, A. Cheung, L. Grella, M. McCord, H. Percy, K. Standiford, and M. Zywno, “REBL nanowriter: Reflective Electron Beam Lithography,” *Proc. SPIE*, vol. 7271, Alternative Lithographic Technologies, 727107, Mar. 2009, doi: 10.1117/12.817319.

[3] Ming-Shing Sua, Kuen-Yu Tsaia, Yi-Chang Lua, Yu-Hsuan Kuoa, Ting-Hang Peia, and Jia-Yush Yenb, “Architecture for next generation massively parallel maskless lithography system (MPML2),” *Proc. SPIE*, Vol. 7637, Alternative Lithographic Technologies II, 76371Q, Apr. 2010, doi: 10.1117/12.846444.

[4] V. Dai, “Data Compression for Maskless Lithography Systems: Architecture, Algorithms and Implementation,” Ph.D. dissertation, University of California, Dept. Electrical Engineering Computer Science, Berkeley, CA, USA, 2008.

[5] Jeehong Yang, “Lossless Circuit Layout Image Compression Algorithm for Multiple Electron Beam Direct Write Lithography Systems,” Ph.D. dissertation, University of Michigan, 2012.

[6] Cheng-Chi Wu, Jensen Yang, Wen-Chuan Wang, Shy-Jay Lin, “An Instruction-based High-Throughput Lossless Decompression Algorithm for E-Beam Direct-Write System,” *Proc. SPIE*, vol. 9423, Alternative Lithographic Technologies VII, 94231P, Mar. 2015, doi: 10.1117/12.2085278.

[7] Jacob Ziv, Abraham Lempel, "A Universal Algorithm for Sequential Data Compression". *IEEE Transaction on Information Theory*, Vol. IT-23, No. 3, May

1977, pp. 337-343.

[8] Laung-Terng Wang, Yao-Wen Chang, and Kwang-Ting Cheng, "Electronic Design Automation: Synthesis, Verification, and Test (Systems on Silicon)," 1st Edition, *Elsevier Inc.*, 2009, Chapter 12, pp. 687-749.

[9] Lee, C.Y., and Whippany, N. J., "An Algorithm for Path Connections and Its Applications," *IEEE IRE Transactions on Electronic Computers*, Vol. EC-10, No. 3, Sept. 1961, pp. 346-365.

[10] Peter E. Hart, Nils J. Nilsson, Bertram Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, Vol. 4, No. 2, July, 1968, pp. 100-107.

[11] Minsik Cho, Yongchan Ban, and David Z. Pan, "Double Patterning Technology Friendly Detailed Routing," in *IEEE/ACM International Conference on Computer-Aided Design*, 2008, pp. 506-511.

[12] Shao-Yun Fang, "Lithography Optimization for Sub-22 Nanometer Technologies," Ph.D. dissertation, National Taiwan University, Graduate Institute of Electronics Engineering, Taipei, Taiwan, 2013.

[13] Jhih-Rong Gao, and David Z. Pan, "Flexible self-aligned double patterning aware detailed routing with prescribed layout planning," *Proceedings of the ACM international symposium on International Symposium on Physical Design*, 2012, pp. 25-32.

[14] Soukup, J., "Fast Maze Router," *15th Design Automation Conference*, 1978, pp. 100-102.

[15] Akers, Sheldon B., "A Modification of Lee's Path Connection Algorithm," *IEEE Transactions on Electronic Computers*, Vol. EC-16, No. 1, Feb., 1967, pp. 97-98.

[16] 2013 International Technology Roadmap for Semiconductors: http://www.itrs.net/

[17] S.-Y. Lee and B. D. Cook, “PYRAMID-A Hierarchical, Rule-Based Approach Toward Proximity Effect Correction-Part I: Exposure Estimation,” *IEEE Transactions on Semiconductor Manufacturing*, Vol.11, No. 1, 1998, pp. 108-116.

[18] Shy-Jay Lin, Pei-Yi Liu, Cheng-Hung Chen, Wen-Chuan Wang, Jaw-Jung Shin, Burn J. Lin, “Influence of Data Volume and EPC on Process Window in Massively Parallel E-Beam Direct Write,” *Proc. SPIE*, vol. 8680, Alternative Lithographic Technologies V, 86801C,March, 2013, doi: 10.1117/12.2010865.