論文 / 著書情報
Article / Book Information

題目(和文)	VLSI物理設計におけるギャップチャネル配線
Title(English)	Gap Channel Routing in VLSI Physical Design
著者(和文)	下田将之
Author(English)	Masayuki Shimoda
出典(和文)	学位:博士(工学), 学位授与機関:東京科学大学, 報告番号:甲第13号, 授与年月日:2024年12月31日, 学位の種別:課程博士, 審査員:髙橋 篤司,一色 剛,佐々木 広,藤木 大地,ISLAM A K M MAHFUZUL
Citation(English)	Degree:Doctor (Engineering), Conferring organization: Institute of Science Tokyo, Report number:甲第13号, Conferred date:2024/12/31, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Gap Channel Routing in VLSI Physical Design

Masayuki Shimoda

Advisor: Professor Atsushi Takahashi

Department of Information and Communications Engineering
School of Engineering
Institute of Science Tokyo

December 2024

Abstract

Routing has a significant impact on chip design and requires dedicated effort in each design style and situation.

With recent technological advances for 3D integration, some recent 3D IC designs need to realize high connectivity demand in one direction, assumed horizontal direction in the dissertation, by utilizing limited routing resources. In the routing layer, there are obstacles laid out in a 2D array, and the routing areas span horizontally between obstacles. It is preferred that routing nets to the areas is completed by a single-trunk Steiner tree. Routing trunks of nets with various inherent widths by given limited areas needs efficient use of the areas.

An objective of the routing problem is to accomplish the routing to minimize routing resources used: especially, routing areas and wirelength. If routing areas cannot be used efficiently, many unnecessary routing areas would be used, resulting in fault of routing with given routing areas. The routing areas should be used efficiently to complete routing, and the remaining routing area can be utilized to improve the quality of chip design such as the total wirelength. A shorter wirelength reduces signal delay and power consumption, and thus the wirelength reduction is crucial for improving the performance and efficiency of the chip. Therefore, while considering routing design constraints, the minimization of routing areas and wirelength plays a key role in the advanced chip design.

Routing algorithms to minimize routing resources have been investigated over several decades. However, existing algorithms cannot achieve efficient routing solutions in the scenario aforementioned because they do not assume separated routing areas and that pins are scattered throughout the chip. Also, their solutions are complicated by dog-leg routing. To realize a high-performance chip design, research and develop algorithms to obtain the efficient solutions for the scenario are necessary.

The dissertation proposes greedy heuristic algorithms to minimize routing areas and wirelength for the routing problem. Their proposed heuristic algorithms allocate trunks of nets to gaps on a gap-by-gap basis. The net priority, or allocation order of trunks of nets, affects the total routing areas used as well as the total wirelength. Since there is a trade-off between them, the dissertation develops an algorithm to use as small routing areas as possible first, followed by developing algorithms to minimize wirelength while maintaining the routing areas used.

First, the dissertation models the routing problem as gap channel routing problem (GCR) to standardize and clarify the problem. In GCR, various-width trunks of nets are allocated to separated routing areas, called gaps. The routing areas used correspond to the number of gaps used. The wirelength of a net depends on the y-coordinate at which the trunk of the net is allocated. An optimal allocation of trunks often contains conflicts, and it is impossible to allocate all trunks to their optimal gaps in general since each gap has a limited capacity. For

obtaining efficient solutions, it is necessary to allocate the trunks of nets while considering other trunks so that as many trunks as possible are routed to the optimal gap.

Second, the dissertation proposes ceiling and packing algorithm (CAP) to minimize the number of gaps used. CAP allocates the trunk of a net repeatedly so that each gap is filled as much as possible by adopting an appropriate order of allocation. CAP gives a higher priority to wider trunks in allocation as first-fit-decreasing adopts in Bin-Packing while the allocation order changes flexibly based on allocated and unallocated trunks of nets. The experimental results show that CAP achieves the least number of gaps used compared to conventional channel routing algorithms, Left-Edge and NLEA.

Finally, the dissertation proposes two algorithms: criticality-based ceiling and packing algorithm (CCAP) and Gap Swap-Flip (GSF) to minimize the wirelength with as small number of gaps used as possible.

CCAP is a greedy heuristic algorithm that is conscious of congested gaps and determines the order of gaps and nets so that as many trunks of nets as possible are allocated with the smaller wirelength as much as possible. CCAP is based on CAP and integrated congestion-aware gap order (CGO) and criticality-based net priority. CGO determines the order of the gaps to be routed so that as many nets as possible can be routed in small wirelength. Criticality-based net priority is priority to routing a net whose wirelength will be smaller if it is routed to the currently processing gap than to the remaining gaps. Incorporating them with CAP reduces the wirelength significantly while minimizing the increase in the number of gaps used. The evaluation results show that CCAP significantly reduced the wirelength by using almost the same number of gaps as Left-Edge and CAP.

GSF is a post-processing algorithm, where given an initial allocation, GSF first reduces the wirelength outside gaps by swapping the allocation between two gaps and then reduces the wirelength inside a gap by reversing the allocation within the gap. It is possible to shorten the wirelength as much as possible without changing the number of gaps used.

Through the dissertation, their proposals obtain an efficient routing solution for GCR that is necessary to realize a high-performance chip design. Thus the dissertation contributes to these advanced chip designs.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my supervisor, Professor Atsushi Takahashi, for their invaluable guidance, support, and encouragement throughout my research. Their expertise and insights have significantly contributed to this work, and their patience and motivation have inspired me to grow both academically and personally.

I am also deeply grateful to my co-researchers at KIOXIA Corporation, especially Chikaaki Kodama and Kosuke Yanagidaira, for their collaborative efforts, insightful discussions, and technical assistance, which were instrumental in the success of this thesis.

I would like to express my heartfelt gratitude to Hiroki Nakahara, the president of Tokyo Artisan Intelligence Co., Ltd., and my mentor for my bachelor's and master's studies. During my time working at Tokyo Artisan Intelligence Co., Ltd., he graciously adjusted my work schedule to accommodate my academic commitments. Thanks to his understanding and flexibility, I was able to fully dedicate myself to my research. His mentorship and support have been invaluable throughout my academic journey.

I would also like to thank all the former and current members of my research laboratory. To my dear friends and colleagues, especially Akira Jinguji and Zezhong Wang, your companionship and late-night brainstorming sessions kept me motivated, even during the toughest times. The countless coffees and conversations shared will remain some of my fondest memories.

Additionally, I extend my heartfelt appreciation to my family, whose constant support and understanding helped me through the most challenging times of this journey. Especially my wife Ayano, for her unwavering patience, love, and understanding throughout this long journey. Her constant encouragement and presence have been my anchor during difficult times. I also want to thank my dog Kotatsu for being a source of comfort and joy during stressful moments.

Finally, this thesis is not just the culmination of my efforts, but also the result of the support, collaboration, and kindness of many people. I am deeply grateful to all who have been part of this journey, directly or indirectly, and I sincerely appreciate the contribution each of you has made.

Contents

Chapter 1

Introduction

1.1 Motivation

In response to the end of Moore's Law that predicts that the number of transistors on a chip would double every two years [1], the importance of 3D integration has grown [2, 3]. The 3D integration refers to the method of stacking chips vertically to create more compact and efficient electronic devices, leading to improvement in key metrics in evaluating semiconductor technologies: power, performance, area, and cost (PPAC). One of the key challenges in 3D integration is the interconnection technology used to link the stacked chips. It is essential to realize as many connections as possible with high density to achieve fast and large-scale data communication between the stacked chips.

One of the interconnection technology is *hybrid bonding* (HB) [4]. HB is a technology that bonds two wafers directly without the use of any additional intermediate layers, resulting in higher device density and reduced interconnect parasitic [5]. In addition, HB enables finer pitch interconnections (or bonding pad) compared to conventional methods like wire bonding [6, 7] or bump connections [8], significantly increasing data transfer speeds and reducing power consumption.

Due to the advantages, HB has been actively studied by many researchers [9, 10] and has been applied to several applications such as DRAM [11], CMOS image sensors [12], 3D NAND flash [13–15]. Some of CMOS image sensors [12] and 3D NAND flash memory [13–15] are already commercialized since HB can perform in a conventional semiconductor wafer production line.

Let is take advanced 3D NAND flash memory [13, 14] as an example. The 3D NAND flash memory is manufactured by bonding a memory cell array wafer that stores information and a CMOS wafer that includes peripheral circuits for the memory cells. By using HB, the NAND I/O speed can be increased, and the memory density can be also increased, which reduces manufacturing costs [13, 14].

Fig. 1.1 shows a top view of a part of the CMOS circuit in the 3D NAND flash [13, 14]. Bonding pads (black rectangles) are laid out in a 2D array on the surface of the wafer. In some layers, the locations under the bonding pads are considered obstacles during the routing phase for the CMOS circuit design in order to keep spaces for signal wires, control circuits, etc. The peripheral circuits (white rectangles) are placed avoiding the obstacles, and they are

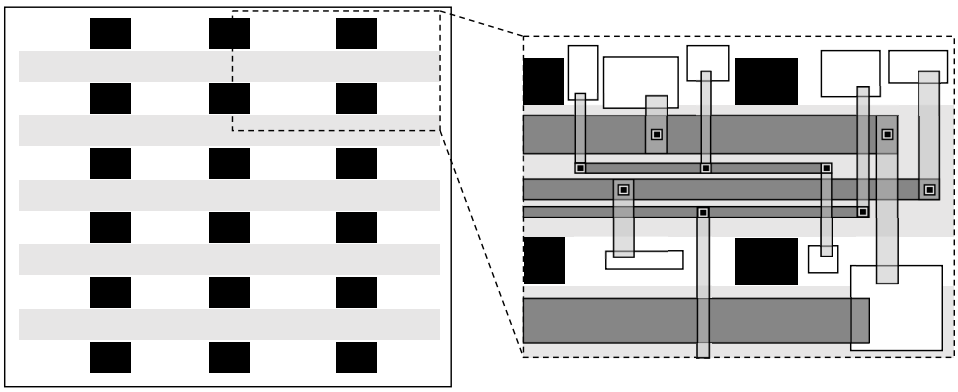


Figure 1.1: CMOS circuit in the 3D NAND flash memory [13, 14]. Routing areas (light gray) are defined horizontally between bonding pads (black). The pins of nets are defined on peripheral circuits (while rectangles). The width of horizontal wires (dark gray) of nets varies.

preferred to be connected by single-trunk Steiner tree with various width. Horizontal wires, or trunks, are routed to the routing areas horizontally spanning between the obstacles (light-gray rectangles), and vertical wires are assumed to be routed in routing layers other than the routing layer used for the trunks. The layers without obstacles are occupied by large circuits such as I/Os, leaving no room for routing. Adding additional routing layers should be avoided as it increases manufacturing costs.

The routing situation and the connection constraint make it hard to use the routing resources efficiently, especially routing areas. Routing algorithms to minimize routing resources have been investigated over several decades [16]. However, the channel routing problem assumed by conventional methods is different from the situation described above; the pins of nets exist at any location in the chip, the existence of multiple candidate routing areas, and routing solutions do not use dog-leg. Due to the difference in the assumed problem setup, existing algorithms are not able to obtain efficient routing solutions in the assumed situation.

The inefficient use of routing resources, especially the routing area and the wirelength, is a major obstacle for realizing gain of PPAC.

If multiple routing areas cannot be used efficiently, many unnecessary routing areas would be used, resulting in fault of routing with given routing areas. When routability issues occur, designers must fix their floor-planning and placement, implying added expense and schedule delay. Routing feasibility check early in the design cycle is important to avoid their costs. Even if routing is possible, the use of unnecessary routing areas limits other optimizations such as wirelength. This may lead to lost opportunities to improve the quality of the chip design.

In addition, minimizing the routing area alone is not enough. Even if the routing area used is small, routing solutions with long wirelength tend to result in more power consumption and congestion. Longer wires consume more power due to increased capacitance, leading to higher dynamic power consumption during signal switching. Also, longer wires require more routing areas, which increases the demand for routing resources in specific regions. This can lead to congestion that constrains the flexibility of routing to optimize secondary objective such as via count and crosstalk.

In summary, research and develop algorithms to obtain the efficient routing solutions for the assumed situation are necessary to realize a high-performance chip design.

1.2 Thesis Contributions

An objective of the routing problem in the thesis is to accomplish the routing to minimize routing resources used: especially, routing areas and wirelength. First, the thesis models the routing problem of the advanced 3D chip design mentioned above as gap channel routing (GCR). The thesis also proposes routing algorithms that minimize routing area and wirelength for GCR. The key contributions are outlined as follows:

- The routing problem in chip development by HB is modeled as GCR to standardize and clarify the problem. In GCR, various-width trunks of nets are allocated to separated routing areas, called gaps, without vertical constraint. The total routing area corresponds to the number of gaps used. The wirelength of a net depends on the y-coordinate at which the trunk of the net is allocated in a gap.
- To minimize the number of gaps used, ceiling-and-packing algorithm (CAP) is proposed for GCR. CAP allocates the trunk of a net repeatedly so that each gap is filled as much as possible by adopting an appropriate order of allocation. The evaluation results show that CAP achieves the least number of gaps compared to Left-Edge [17]-based one and NLEA [18].
- To minimize the vertical wirelength, criticality-based CAP (CCAP) is proposed for GCR. CCAP is based on CAP and adopts congestion-aware gap order (CGO) and criticality-based net priority. Incorporating them into CAP reduces the vertical wirelength significantly while completing the routing by using gaps required for CAP in most cases. Experimental results show that CCAP significantly reduced the wirelength by using almost the same number of gaps as CAP.
- Gap Swap-Flip (GSF) is also proposed to minimize the wirelength from initial allocation. Given an initial allocation, GSF first reduces the wirelength outside gaps by swapping the allocation between two gaps and then reduces the wirelength inside a gap by reversing the allocation within the gap. The wirelength is further reduced while keeping the same routing area used as the initial allocation. The evaluation results show that adapting GSF to the three allocations by Left-Edge, CAP, and CCAP reduces wirelength.

1.3 Thesis Organization

The rest of this thesis is organized as follows.

Chapter 2 reviews the related works that have been made over decades in channel routing and discusses assumed routing situation and algorithms.

Chapter 3 models the routing problem of the advanced 3D chip as gap channel routing (GCR), where it is a problem of allocating net trunks. From the definition of each term, the design rule and the difficulty of the GCR is explained.

Chapter 4 proposes a greedy heuristic algorithm, ceiling and packing algorithm (CAP), to minimize routing area used. CAP allocates the trunk of a net repeatedly so that each gap is filled as much as possible by adopting an appropriate order of allocation. The content of this chapter is largely based on the author's published work [19].

Chapter 5 proposes two algorithms: criticality-based ceiling and packing algorithm (CCAP) that extends CAP proposed in Chapter 4 and gap swap-flip algorithm, to minimize wirelength. The content of this chapter is largely based on the author's published works [20, 21].

Chapter 6 concludes the thesis and discusses the future work.

Chapter 2

Related Works

Gap channel routing (GCR), routing problem of allocating trunks of nets to gaps, is similar to a conventional grid base channel routing where pins of nets are located on the boundary of a single rectangle routing region without obstacles. The channel routing [16, 22, 23] is one of the important processes in automated layout design, especially for cell-based design, that significantly affects the quality of the chip. Various algorithms for the conventional channel have been proposed so far [17, 23–26], but they cannot be directly applied to GCR. For example, a popular channel routing algorithm Left-Edge [17] neither considers the divided routing regions (gaps) nor takes the widths of wires into account. The variety of wire widths of nets needed inhibits the efficient use of gaps, and Left-Edge cannot obtain reasonable solutions in GCR.

Gridless channel routing [18, 27–33] has also been a subject of research for a long time in conventional channels. However, these works are targeted to apply to problems on a single channel, whereas the target of this paper is a channel that is divided into fixed-width gaps, and thus these works cannot work well.

Bottleneck channels where several adjacent layers are used for connections in the direction were discussed in [34] to meet high connectivity demand in one direction. Similarly, generalized channels where the connection of each net is restricted as a single-trunk Steiner tree to minimize the consumption of routing resources in the direction was discussed in [35]. However, they handle unit-width wires in a grid-based routing area.

Area routers [36, 37] that prefer to use multiple horizontal segments do not fit the situation where routing each net by a single trunk is required. Routing algorithms such as negotiation-based routers [38, 39] as well as machine learning-based routers [40, 41] cannot be used in practical design scenes so far due to a large computation time required. In the practical design scene, engineering change orders (ECOs) happen frequently, and a short computation time to respond to ECO is essential.

Chapter 3

Gap Channel Routing

3.1 Gap Channel

A routing layer that contains regularly placed obstacles with high horizontal connectivity demand is modeled as a *gap-channel*. The gap-channel is a single-layer rectangle routing area that consists of a set of *gaps* where horizontal trunks of nets are allocated, as shown in Fig. 3.1.

The coordinate of the lower left corner of a gap-channel is defined as $(0,0)$. The (vertical) width and (horizontal) length of the gap-channel are denoted by w_c and l_c, respectively. A gap g in the gap-channel spans the gap-channel horizontally. The y-coordinate of the lower boundary of gap g is given by $y(g)$, and the location of g is specified by its lower left corner which is given by $(0, y(g))$. The (vertical) width of gap g is denoted by w_g, and the (horizontal) length of g is equal to l_c. Gaps in the gap-channel do not overlap each other. The set of gaps in the gap-channel is given by $G_{in} = \{g_i\}_{i=1}^{k}$ where k is the number of gaps and $0 \leq y(g_1) < y(g_2) < \cdots < y(g_k)$. Note that $y(g_i) + w_g < y(g_{i+1})$ for any i $(1 \leq i < k)$ and that $y(g_k) + w_g \leq w_c$.

As an input of the gap-channel routing problem, the set of nets $N_{in} = \{n_i\}_{i=1}^{m}$ is given where m is the number of nets. A net n consists of $r(n)$ pins, that is, $n = \{p_i\}_{i=1}^{r(n)}$. The coordinate of pin p is denoted by $(x(p), y(p))$. The pins of nets exist at any location inside the channel unlike conventional channel routing where pins exist only on the boundary of the channel.

All the pins of a net are required to be connected by a single trunk Steiner tree that consists of the horizontal trunk of the net and vertical wires that connect the pins of the net and its trunk. The trunks of nets are allocated to gaps such that they do not overlap. The location of the trunk of a net inside a gap is specified by the offset from the lower boundary of the gap. Vertical wires are routed on routing layers other than the routing layer used for the trunks of nets.

3.2 Trunk Allocation

Let $\mathcal{A}\colon n \mapsto (\mathcal{G}(n), \mathcal{S}(n))$ be the allocation function of trunks of nets to gaps where $\mathcal{G}\colon N_{in} \to G_{in}$ and $\mathcal{S}\colon N_{in} \to [0, w_g]$ are the gap allocation and the offset allocation, respectively. When trunk $T(n)$ of net n is allocated by $\mathcal{A} = (\mathcal{G}, \mathcal{S})$, the y-coordinate of $T(n)$ is given by $y(n, \mathcal{A}) = y(\mathcal{G}(n)) + \mathcal{S}(n)$. The y-coordinate $y(n, \mathcal{A})$ and the offset $\mathcal{S}(n)$ of net n by $\mathcal{A}$ are also denoted by $y(n)$ and $s(n)$, respectively, for simplicity.

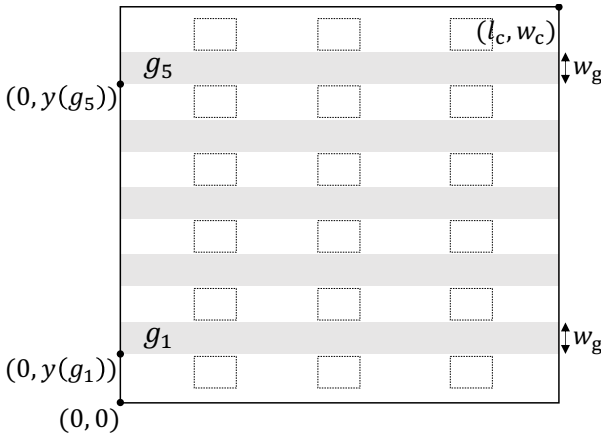


Figure 3.1: Gap-channel.

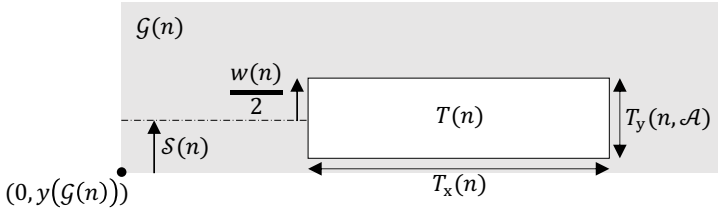


Figure 3.2: Trunk allocation.

The vertical range of trunk $T(n)$ whose width is $w(n)$ is determined by $\mathcal{A}$ as $T_y(n, \mathcal{A}) = [y(n) - w(n)/2, y(n) + w(n)/2]$. While, the horizontal range of trunk $T(n)$, which is independent of the allocation, is defined as $T_x(n) = [x_{min}(n), x_{max}(n)]$ where $x_{min}(n)$ and $x_{max}(n)$ are the leftmost and the rightmost x-coordinates of pins in n, respectively.

In a feasible allocation, for any distinct pair of trunks, the intersection of the vertical range of them is empty. For any distinct pair of nets n and n' in N_{in} in feasible allocation $\mathcal{A}$, $T_x(n) \cap T_x(n') = \emptyset$ is satisfied.

In summary, GCR is to allocate trunks of nets to gaps so that they meet design requirement: arbitrary two trunks are not overlapped, and all trunks fit within the GAP. For given a net set N_{in} and a gap set G_{in}, GCR is represented by

Gap Channel Routing (GCR)

Instance: Net set N_{in}, Gap set G_{in}.

Output: Allocation $\mathcal{A} = (\mathcal{G}, \mathcal{S})$: $N_{in} \rightarrow \left(G_{in}, [0, w_g]\right)$.

Constraints:

- $\mathcal{G}(n) \in G_{in}, \forall n \in N_{in}$,
- $w(n)/2 \leq \mathcal{S}(n) \leq w_g - w(n)/2, \forall n \in N_{in}$,
- $T_x(n) \cap T_x(n') = \emptyset$ or $T_y(n, \mathcal{A}) \cap T_y(n', \mathcal{A}) = \emptyset, \forall n, n' \in N_{in}$.

The main objective of GCR in the dissertation is to minimize the number of gaps used.

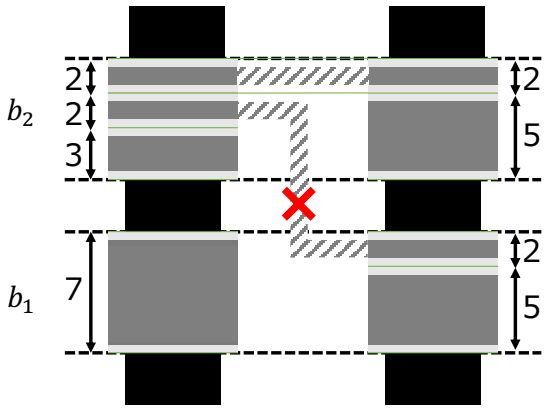


Figure 3.3: An enhanced Bin Packing derived from GCR.

3.3 Difficulty

GCR where the objective is to minimize the number of gaps used is equivalent to Bin Packing problem when all trunks horizontally overlap with each other. There exists a trivial transformation from the decision version of Bin Packing to the decision version of GCR. Bin packing is known to be NP-hard in general [42]. Therefore, GCR is also NP-hard in general, and it is difficult to obtain an optimal solution efficiently in general.

Formally, Bin Packing is defined as follows:

BIN PACKING (BP)

Instance: Finite set U of items, a size $s(u) \in \mathcal{Z}^+$ for each $u \in U$, a positive integer bin capacity B, and a positive integer K.

Question: Is there a partition of U into disjoint sets $U_1, U_2, \ldots, U_K$ such that the sum of the sizes of the items in each U_i is B or less?

Even though BP is NP-complete in the strong sense in general, it is solvable in polynomial time by exhaustive search if bin capacity is fixed [42]. However, exhaustive search is impractical even for a small bin capacity when the number of items is large. Therefore, various heuristics have been proposed for BP [43].

One of the heuristics for BP is First-Fit Decreasing (FFD). Algorithm 1 gives the pseudo code of FFD. FFD allocates an item in descending order of size to the first fit bin to be found. FFD is a heuristics, and does not necessarily find a good allocation. For example, FFD fails to obtain an optimal allocation when the multi-set $\{3,3,3,3,2,2,2,2\}$ is given as item sizes and bin size $B = 10$ [44, 45].

Examples of bin packing where bin size $B = 7$ are given in Fig. 3.3. There are two sets of items whose sizes are given as multi-sets $\{7,3,2,2\}$ and $\{5,5,2,2\}$ where items in each set are allocated to a bin in its corresponding set of bins. Let us assume that each set corresponds to the set of widths of trunks to be allocated to a portion of the channel in GCR. FFD packs them successfully as $b_1 = \{7\}$, $b_2 = \{3,2,2\}$ and $b_1 = \{5,2\}$, $b_2 = \{5,2\}$, respectively. On the other hand, in GCR, a trunk (item) has to be allocated to bins with the same index if it spans multiple portions of the channel. In case that two items with size 2 span two portion of the channel, it

Algorithm 1 First-Fit-Decreasing (FFD)

Require: set of items U, set of bins G
1: $U \leftarrow$ DescendingWidthSort(U)
2: **for all** $b \leftarrow G$ **do**
3: $b \leftarrow \emptyset$
4: **end for**
5: **while** $U \neq \emptyset$ **do** ▷ descending order of size
6: $n \leftarrow$ delete(U)
7: **for all** $b \leftarrow G$ **do** ▷ search fit bin
8: **if** $w(n) + \sum_{n' \in b} w(n') \geq w(b)$ **then**
9: **continue**
10: **end if**
11: $b \leftarrow b \cup \{n\}$ ▷ allocate n to the first fit bin b
12: **end for**
13: **end while**

is impossible to allocate them in two gaps. Thus, GCR is more complicated because GCR is a problem of accommodating 2-dimensional items with x-coordinates instead of 1-dimensional items. Therefore, an algorithm to find a better solution efficiently is required.

Chapter 4

Gap Channel Routing to Minimize Routing Area

4.1 Introduction

This chapter deals with the problem of minimizing the number of gaps used in GCR. Formally, the problem is defined as follows:

Gap Channel Routing to Minimize Routing Area

Instance: Net set N_{in}, Gap set G_{in}.

Output: Allocation $\mathcal{A} = (\mathcal{G}, \mathcal{S})$: $N_{\text{in}} \rightarrow \left(G_{\text{in}}, [0, w_{\text{g}}]\right)$.

Objective: Minimization of the number of gaps used $|\{\mathcal{G}(n) \mid n \in N_{\text{in}}\}|$.

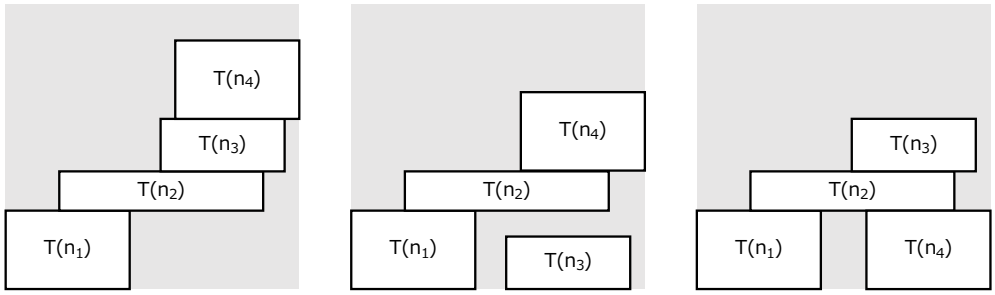
Constraints:

- $\mathcal{G}(n) \in G_{\text{in}}, \forall n \in N_{\text{in}}$,
- $w(n)/2 \leq \mathcal{S}(n) \leq w_{\text{g}} - w(n)/2, \forall n \in N_{\text{in}}$,
- $T_{\text{x}}(n) \cap T_{\text{x}}(n') = \emptyset$ or $T_{\text{y}}(n, \mathcal{A}) \cap T_{\text{y}}(n', \mathcal{A}) = \emptyset, \forall n, n' \in N_{\text{in}}$.

It is unnecessary to use a small number of gaps when sufficient number of gaps is given as inputs. However, this objective function enables us to evaluate the performance of algorithms as well as the validity of chip design. Also, unused gaps can be utilized to improve the quality of chip design such as the total vertical wirelength.

4.2 Trunk Allocation Scheme

GCR is a problem of allocating trunks on a two-dimensional gaps without overlap.Unlike conventional channel routing, tracks are not defined, so it is necessary to decide at which height of which gap to allocate trunks.



(a) (n_1, n_2, n_3, n_4) **(b)** (n_1, n_3, n_2, n_4) **(c)** (n_1, n_4, n_2, n_3)

Figure 4.1: Trunk allocations by different orders.

Proposed algorithm allocates trunks to a gap by gap. For each gap, it allocates a trunk repeatedly as low as possible while avoiding jumping over other trunks allocated so far when it is dropped from above.

Formally, the set of nets in N whose trunks contain x is denoted by $M(x,N) = \{n \in N \mid x \in I_{\mathrm{x}}(n)\}$, and the set of nets in N whose trunks horizontally overlap with trunk $T(n)$ of net n is denoted by $M(n,N) = \{n' \in N \mid I_{\mathrm{x}}(n') \cap I_{\mathrm{x}}(n) \neq \emptyset\}$. The y-coordinate of the bottom side of trunk $T(n)$ is $\max_{n' \in M(n,N)} (\mathcal{S}(n') + w(n')/2)$ when the set of trunks $T(N)$ have been allocated to a gap and the trunk $T(n)$ is to be allocated to the gap.

Fig. 4.1 gives examples of allocation results of the set of four trunks $T(N_{\mathrm{in}}) = \{T(n_i)\}_{i=1}^{4}$ in different allocation orders. In Fig. 4.1(a), $T(n_3)$ is allocated on the top of $T(n_2)$ even if there is enough space below $T(n_2)$. This is not an optimal in terms of width used in the allocation. An optimal result is shown in Fig. 4.1(c). Note that there are various optimal results in terms of width used and that allocation orders that derive an optimal result are not unique.

The order of gaps to allocate trunks has no effect on the evaluation of the proposed algorithm. For simplicity, the order from the lowest gap among gaps not used so far is assumed in this chapter, i.e., $g = \arg\min_{g \in G}\ y(g)$ is selected as the gap to allocate, where G is the set of gaps not used so far.

4.3 Baseline: Left-Edge (LE)

The algorithm proposed in this chapter uses Left-Edge (LE) [17] as a baseline. LE is a well-known greedy algorithm for conventional channel routing where horizontal tracks are defined to which trunks are allocated. For each track, LE allocate trunks not allocated so far from left to right with as few gap as possible without vertical constraint violation. LE uses the minimum number of tracks required if there is no vertical constraint. Although LE allocates unit-width trunks to tracks in conventional grid-based channels, tracks are not defined in GCR since various width trunks exist.

In the following, LE refers an extended LE for GCR unless otherwise specified. The pseudo code of LE is given in Algorithm 2.

The iteration of trunk allocation from left to right is referred by *round* in the dissertation. In each round, a trunk whose left end is the leftmost among the trunks not allocated so far is repeatedly allocated by the allocation scheme if no violation occurs. The function "LeftEdgeSort(N_{in})"

sorts trunks based on the left end of trunks. LE skips the allocation of a higher priority trunk if either horizontal constraint (HC) or gap width constraint (WC) is violated. WC is violated if the trunk cannot fit to the gap, and the details of HC is explained the next subsection.

The function "ceiling_width(n, N')", where $T(n)$ is to be allocated when the trunks $T(N')$ have been allocated, gives the y-coordinate of trunk $T(n)$ according to the trunk allocation scheme. The rounds for a gap are repeated until no more trunk is allocated to the gap.

Note that, in GCR, there is no guarantee that LE achieves the minimum number of gaps used in allocation, unlike conventional channel routing for uniform-width wires. The result obtained by LE is shown in Fig. 4.2(a) where the allocation order is $(n_1, n_2, n_3, n_4, n_5, n_6)$. The result obtained by NLEA is shown in Fig. 4.2(b) where the allocation order is $(n_1, n_3, n_5, n_4, n_6, n_2)$. An optimal result in terms of the width used is shown in Fig. 4.2(c).

4.3.1 Horizontal Constraint (HC)

HC makes sure that all trunks allocated in the same round do not horizontally overlap each other. In each round, trunks are allocated from left to right with no overlap because allocating trunks that overlap each other in a round would result in wasted space (like the allocation of $T(n_1)$ and $T(n_2)$ shown in Fig. 4.1(a)).

The trunk of a net violates HC if the left end of the trunk is to the left of the rightmost end of trunks allocated in the round. This is checked by step 9 in LE where $x_{\min}(n)$ and x correspond to the left end of the trunk and the rightmost end of trunks allocated in the round, respectively. If HC is violated, allocation of a higher priority net is skipped. Even though the trunk is not actually overlapped with trunks allocated in the round, it is skipped if HC is violated. By introducing HC, it is guaranteed that no overlap among trunks allocated in a round occurs by a simple checking.

4.4 Ceiling and Packing Algorithm (CAP)

This chapter proposes Ceiling-and-Packing algorithm (CAP) to use gaps efficiently in GCR. CAP gives a higher priority to wider trunks in allocation as First-Fit Decreasing (FFD) adopts in Bin-Packing as mentioned in Chapter 3. CAP allocates trunk repeatedly as LE does while trying to cover the maximum density zone as much as possible during allocation. Also, CAP adopts ceiling during allocation to allocate trunks as low as possible.

The pseudo code of CAP is given in Algorithm 3. The function "CAPSort(N_{in})" sorts trunks according to the width of trunk, and then ties are broken by the leftmost principal. CAP skips allocation of a higher priority trunk if any of the following constraints is violated: HC, zone constraints (ZC), or ceiling constraint (CC). The details of ZC and CC are explained in the following subsections.

The result obtained by CAP is shown in Fig. 4.2(c). CAP prioritizes the trunks of nets in order of $(n_2, n_6, n_3, n_4, n_5, n_1)$ while the allocation order is $(n_6, n_2, n_5, n_1, n_3, n_4)$ due to the three constraints.

Algorithm 2 Left-Edge (LE) for GCR.

Require: set of nets N_{in}, set of gaps G_{in}

$N \leftarrow \text{LeftEdgeSort}(N_{\text{in}})$, $G \leftarrow G_{\text{in}}$
while $N \neq \emptyset$ **do** ▷ next gap
  $g \leftarrow \text{delete}(G)$
  $N' \leftarrow \emptyset$, $f \leftarrow \top$
  while $f = \top$ **do** ▷ next round
    $x \leftarrow -\infty$, $U \leftarrow N$, $f \leftarrow \mathsf{F}$
    while $U \neq \emptyset$ **do** ▷ next allocation
      $n \leftarrow \text{delete}(U)$
      if $x_{\min}(n) \leq x$ **then** ▷ violate HC
        continue
      end if
      $y \leftarrow \text{ceiling\_width}(n, N')$
      if $y + w(n) > w_{\text{g}}$ **then** ▷ violate WC
        continue
      end if
      $a(n) \leftarrow (g, y)$, $N' \leftarrow N' \cup \{n\}$ ▷ allocate n to g
      $x \leftarrow x_{\max}(n)$, $f \leftarrow \top$
      $N \leftarrow \text{delete}(n, N)$
    end while
  end while
end while

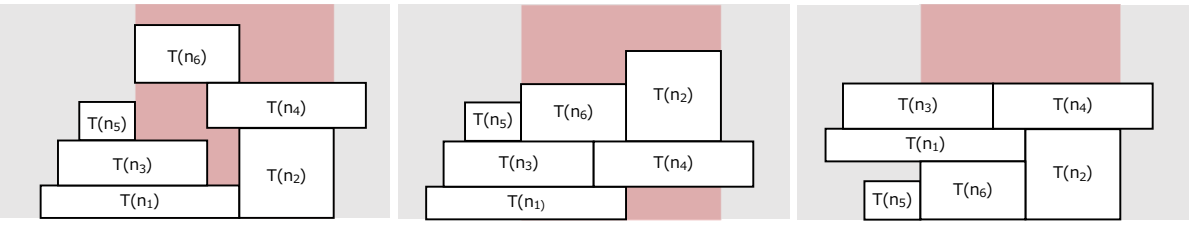
4.4.1 Zone Constraint (ZC)

CAP considers the maximum density zones where the demand for the routing resources to allocate trunks of the remaining nets are the highest.

Formally, *density* of set of nets N at x is the sum of widths of trunks of nets N whose horizontal range contain x and is denoted by $d(x, N) = \sum_{n \in \{n \in N | x \in T_{\text{x}}(n)\}} w(n)$. Then, the maximum density zone of nets $Z(N)$ is defined as $Z(N) = \{x \mid d(x, N) = D(N)\}$, where $D(N)$ is the maximum density denoted by $D(N) = \max_{x \in [0, l_{\text{c}}]} d(x, N)$. For a set of trunks $T(N)$ not allocated so far, the maximum density of nets N gives a lower bound of the width of the channel that are used for allocation of $T(N)$.

In each round, CAP allocates trunks so that this lower bound is reduced as much as possible. At the beginning of each round, CAP finds the maximum density zone of nets not allocated so far. CAP allocates a trunk only when all the maximum density zone to the left of the trunk is covered by trunks allocated in the round so far. CAP skips the allocation of a trunk if there is the maximum density zone to the left of the trunk that is not covered by trunks allocated in the round, that is, if zone constraint (ZC) is violated.

For example, in Fig. 4.2(c), the priority of trunks used in CAP is $(n_2, n_6, n_3, n_4, n_5, n_1)$ where the first criteria is the the descending order of widths of trunks. At the beginning, the maximum



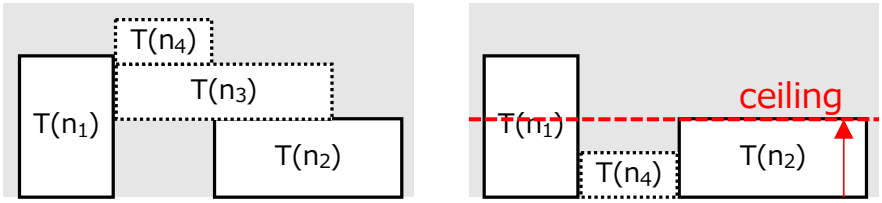
(a) LE $(n_1, n_2, n_3, n_4, n_5, n_6)$ **(b)** NLEA $(n_1, n_3, n_5, n_4, n_6, n_2)$ **(c)** CAP $(n_6, n_2, n_5, n_1, n_3, n_4)$

Figure 4.2: Examples of allocations. Red zone indicates the maximum density zone of N_{in}.

Algorithm 3 Ceiling-and-Packing (CAP)

Require: set of nets N_{in}, set of gaps G_{in}

$N \leftarrow \text{CAPSort}(N_{\text{in}})$, $G \leftarrow G_{\text{in}}$
while $N \neq \emptyset$ **do** ▷ next gap
  $g \leftarrow \text{delete}(G)$
  $N' \leftarrow \emptyset$, $C \leftarrow (w_g)$
  while $C \neq \emptyset$ **do** ▷ next round
    $Z \leftarrow \{x \mid d(x, N) = D(N)\}$
    $x \leftarrow -\infty$, $U \leftarrow N$, $c \leftarrow \text{top}(C)$
    while $U \neq \emptyset$ **do** ▷ next allocation
      $n \leftarrow \text{delete}(U)$
      if $x_{\min}(n) \leq x$ **then** ▷ violate HC
        continue
      end if
      if $(x, x_{\min}(n)) \cap Z \neq \emptyset$ **then** ▷ violate ZC
        continue
      end if
      $y \leftarrow \text{ceiling\_width}(n, N')$
      if $y + w(n) > c$ **then** ▷ violate CC
        continue
      end if
      $a(n) \leftarrow (g, y)$, $N' \leftarrow N' \cup \{n\}$ ▷ allocate n to g
      $x \leftarrow x_{\max}(n)$
      $C \leftarrow \text{insert}(y + w(n), C)$
      $N \leftarrow \text{delete}(n, N)$, $U \leftarrow N$
    end while
    if $x = -\infty$ **then** ▷ no allocation in round
      $C \leftarrow \text{delete}(c, C)$
    end if
  end while
end while



(a) No CC **(b)** $T(n_4)$ meets CC

Figure 4.3: Ceiling constraint (CC).

density zone ranges from the left end of $T(n_6)$ to the right end of $T(n_2)$. In the first round, CAP tries to allocate $T(n_2)$ first, but it is skipped since the maximum density zone to the left of $T(n_2)$ that is not covered exists. CAP allocates $T(n_6)$ first, then allocates $T(n_2)$ second, and terminates the first round.

4.4.2 Ceiling Constraint (CC)

CAP sets to the ceiling of allocation in each round to pack as many trunks as possible into a low location. A ceiling gives the upper limit of y-coordinate of trunks to be allocated in the round. CAP skips the allocation of a trunk if the location of the trunk exceeds the ceiling when the trunk is allocated according to the allocation scheme, that is, if ceiling constraint (CC) is violated.

In each round, the minimum y-coordinate among ceiling candidates is selected as the ceiling. A candidate of ceiling is the location of the top boundary of each trunk allocated. A candidate is selected as the ceiling of a round among the set of candidates C, and it is removed from C if nothing is allocated in the round. When a trunk is allocated, the new candidate is inserted to C, but C does not contain multiple identical y-coordinates. The function "insert$(y+w(n),C)$" maintains C when a trunk is allocated. At the first round and the final round of a gap, the ceiling is set to the width of the gap.

Fig. 4.3 illustrates the effects of CC. Assume that the priority of trunks is given as (n_1,n_2,n_3,n_4), and that the trunks $T(n_1)$ and $T(n_2)$ have already been allocated, but $T(n_3)$ and $T(n_4)$ are not allocated yet. If CC is not enforced, the allocation shown in Fig. 4.3(a) is obtained where $T(n_4)$ is allocated above $T(n_3)$. While, if the top boundary of $T(n_2)$ is set to the ceiling, $T(n_3)$ violates CC, and $T(n_4)$ is allocated before $T(n_3)$ as shown in Fig. 4.3(b). CC prioritizes a trunk that can be allocated below the ceiling, enabling us to utilize the routing area effectively.

4.5 Experimental Results

In order to evaluate the proposed algorithm, three types of benchmarks that consist of trunks of four different widths in different probabilities are prepared. They are used as inputs for multiple gaps as well as for a single gap.

The single gap has an infinite width virtually, and the total width is used to evaluate algorithms. In multiple gaps, it is requested to complete the allocation by using given gaps in

Table 4.1: Setting (w indicates trunk width).

scenario (prefix)	Pr(w) $w=1$	$w=2$	$w=3$	$w=4$
Sparse (c1)	0.80	0.10	0.08	0.02
Dense (c2)	0.50	0.30	0.15	0.05
FFD-killer (c3)	0	0.50	0.50	0

Table 4.2: The additional width for $w_g = \infty$ in single gap.

Benchmark name	#net $\lvert N_{in}\rvert$	Density (Lower Bound) $D(N_{in})$	The additional width beyond Density (Lower Bound) LE	NLEA	CAP w/o ZC w/o CC	CAP w/o CC	CAP w/o ZC	CAP (ours)
c1-b1	10	13	0	0	0	0	0	0
c1-b2	100	91	7	23	1	1	0	0
c1-b3	500	377	51	191	10	10	2	2
c1-b4	1,000	716	91	461	14	13	8	3
c2-b1	10	16	0	0	0	0	0	0
c2-b2	100	121	11	29	1	1	0	0
c2-b3	500	497	71	261	4	4	1	1
c2-b4	1,000	941	135	602	17	14	7	5

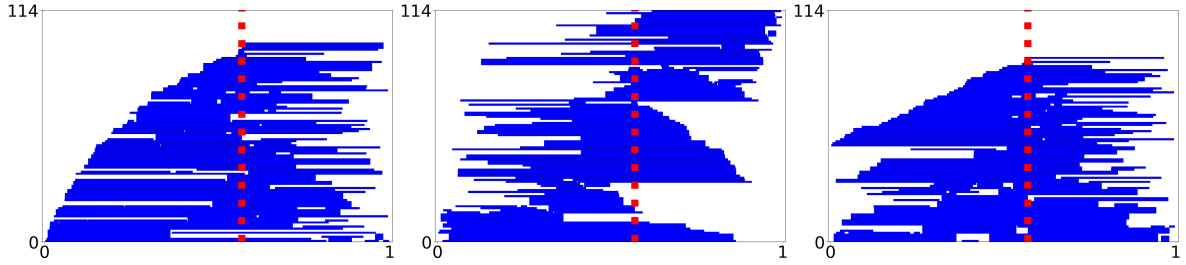
practice. To evaluate algorithms, assuming enough gaps are given as an input, the number of gaps used beyond the lower bound is used as evaluation.

The benchmarks were randomly generated where the minimum and maximum x-coordinates and width of each trunk were defined. The minimum and maximum x-coordinates of each trunk were generated uniformly from range $[0,1]$, that is, $x_{min}, x_{max} \sim \mathcal{U}(0,1)$. The width of each trunk w was selected among 1, 2, 3, and 4 according to the probabilities given in Table 4.1.

For comparisons, LE, NLEA [18], CAP, and derivations of CAP: CAP without allocation skip by ZC and CC, CAP without CC, and CAP without ZC. They were implemented by Python 3.10.9, and executed on Apple M2 CPU. The execution time of LE, NLEA, and CAP was 0.3, 5.2, and 180.1 seconds, respectively, for the largest benchmark.

The results for the single gap are shown in Table 4.2. All algorithms achieve lower bound for 10 trunks. For LE and NLEA, the differences from lower bound increases as the number of trunks increases, and the difference are more than 10% of lower bound. On the other hand, the proposed algorithm achieves lower bound for 100 trunks, and the differences of widths are within 5 for 1,000 trunks.

The results with the benchmark c1-b2 are shown in Fig. 4.4. The red dotted line is drawn in the maximum density zone of N_{in}. It is confirmed that LE (Fig. 4.4(a)) and NLEA (Fig. 4.4(b)) are unable to cover all the maximum density zone, and that need more widths than lower bound to complete the allocation. On the other hand, the proposed CAP (Fig. 4.4(c)) covers all the zone, and the total widths used is the same as lower bound. Even when the routing area is not divided, the proposed CAP completes routing with less total widths compared with



(a) LE ($W = 98$) **(b)** NLEA ($W = 114$) **(c)** CAP (ours) ($W = 91$(optimal)).

Figure 4.4: Results on c1-b2 in single gap. Trunks (blue rectangles) and the maximum density zone (red dotted line).

Table 4.3: The additional gaps for $w_g = 10$ in multiple gaps.

Netlist		Density	Lower	The number of gaps used beyond Lower Bound					
				LE	NLEA	CAP			
name	#net $\lvert N_{in}\rvert$	$D(N_{in})$	Bound $\lceil D(N_{in})/w_g \rceil$			w/o ZC w/o CC	w/o CC	w/o ZC	(ours)
c1-b1	10	13	2	0	0	0	0	0	0
c1-b2	100	91	10	0	2	0	0	0	0
c1-b3	500	377	38	4	23	1	1	0	0
c1-b4	1,000	716	72	8	51	1	1	1	0
c2-b1	10	16	2	0	0	0	0	0	0
c2-b2	100	121	13	0	3	0	0	0	0
c2-b3	500	497	50	5	31	1	0	0	0
c2-b4	1,000	941	95	11	68	1	1	0	0
c3-b1	10	19	2	0	0	0	0	0	0
c3-b2	100	161	17	0	6	1	1	1	1
c3-b3	500	704	71	5	47	5	5	5	5
c3-b4	1,000	1,343	135	11	112	9	9	9	9

conventional ones.

The results for the multiple gaps are shown in Table 4.3. In benchmark settings of c1 and c2, only CAP allocates trunks in the minimum number of gaps.

In the setting of FFD-killer c3, where FFD fails to achieve the minimum number of bins in most cases when the size of a bin is 10, CAP requires more number of gaps than lower bound, and even more than LE in c3-b2. Like FFD, CAP will not be able to find certain combinations of trunks to fit a gap. However, as the number of trunks increases, LE also gradually increases the number of excess gaps, which is larger than that of CAP. Since the amount of trunks in an actual chip design is large, CAP is more appropriate than conventional algorithms.

4.6 Summary

In this chapter, CAP for GCR was proposed to minimize routing area used. CAP uses fewer gaps compared with conventional algorithms by prioritizing wide trunks and filling in the gaps as much as possible. Experimental results show that CAP achieves the least number of gaps compared to LE and NLEA. Since an efficient solution of GCR is necessary to realize a high-performance chip design, CAP contributes to these chip designs.

Chapter 5

Gap Channel Routing to Minimize Wirelength

5.1 Introduction

This chapter deals with GCR of minimizing the wirelength while minimizing the increase of the number of gaps obtained by CAP proposed in the previous chapter.

The wirelength depends on the location where the trunk is allocated, and only the vertical wirelength varies. Formally, the vertical wirelength of net n allocated at y is given by $l(n,y) = \sum_{p \in n} |y - y(p)|$. Since allocation $\mathcal{A}$ decides y, $l(n,y)$ is also denoted by $l(n,\mathcal{A})$.

The problem of GCR to minimize the vertical wirelength is defined as follows:

Gap Channel Routing to Minimize Vertical Wirelength

Instance: Net set N_{in}, Gap set G_{in}.

Output: Allocation $\mathcal{A} = (\mathcal{G}, \mathcal{S})\colon N_{\mathrm{in}} \to \left(G_{\mathrm{in}}, [0, w_{\mathrm{g}}]\right)$.

Objective: The total vertical wirelength $\sum_{n \in N_{\mathrm{in}}} l(n,\mathcal{A})$.

Constraints:

- $\mathcal{G}(n) \in G_{\mathrm{in}}, \forall n \in N_{\mathrm{in}}$,
- $w(n)/2 \leq \mathcal{S}(n) \leq w_{\mathrm{g}} - w(n)/2, \forall n \in N_{\mathrm{in}}$,
- $T_{\mathrm{x}}(n) \cap T_{\mathrm{x}}(n') = \emptyset$ or $T_{\mathrm{y}}(n,\mathcal{A}) \cap T_{\mathrm{y}}(n',\mathcal{A}) = \emptyset, \forall n, n' \in N_{\mathrm{in}}$.

In this chapter, two algorithms are proposed: Section 5.2 proposes criticality-based ceiling and packing algorithm (CCAP) that is enhanced from CAP proposed in the previous chapter, and Section 5.3 proposes gap swap-flip that is a post-processing algorithm.

5.2 Criticality-based Ceiling and Packing Algorithm (CCAP)

This section proposes criticality-based ceiling and packing algorithm (CCAP) to minimize the wirelength in GCR. CCAP is based on CAP and introduces a new gap order and a new net pri-

ority: congestion-aware gap order (CGO) and criticality-based net priority. CGO determines the order of the gaps to be allocated so that as many nets as possible can be allocated in small wirelength. Criticality-based net priority is priority to allocating the trunk of a net whose wirelength will be smaller if it is allocated to the currently processing gap than to the remaining gaps. Incorporating them with CAP reduces the wirelength significantly without increasing the number of gaps used too much.

The pseudo code of CCAP is given in Algorithm 4. First, CGO selects the most uncongested gap among unused gaps, and the selected gap is called *target gap* g_{target} in this paper. Based on the target gap, the function "CriticalitySort(N, g, G)" sorts nets according to the width of the trunks, criticality-based net priority, and then ties are broken by the leftmost principal. After the target gap is determined and nets are ordered, the subsequent processing is the same as CAP; allocating the trunks of nets to the target gap until no trunk can be allocated under HC, ZC, and CC. The details of CGO and criticality-based net priority are explained in the following subsections.

5.2.1 Congestion-aware Gap Order

Congestion-aware Gap Order (CGO) prioritizes uncongested gaps so that as many nets as possible are allocated with small wirelength.

In GCR, vertical wirelength depends on y-coordinate at which the horizontal trunk of a net is allocated. Since the allocation of the horizontal trunk is limited to the inside of gaps, the wirelength is greatly affected by which gap the trunk is allocated. It is necessary to select an appropriate gap to shorten the wirelength. However, it is often impossible to allocate some nets to the optimal gap since each gap has a limited routing capacity. To reduce the total wirelength, a routing method that takes other nets into account is needed so that as many nets as possible can be allocated to their best gaps.

Minimizing the number of gaps used can be achieved by allocating nets from the widest ones, and thus the minimum width nets is done later. In many cases, the minimum width nets have a high percentage of total nets and have a large impact on the total wirelength. It is necessary to allocate wide nets, avoiding the gaps where the minimum width nets are desired to be allocated as much as possible.

This paper proposes CGO that considers the gap congestion based on the minimum width nets. Gap congestion is estimated based on the sum of trunk widths of nets that are allocated to the gap when trunks are allocated to the gap to achieve the smallest wirelength of the net. Use of the gap congestion based on the minimum width nets avoids congested gaps to be allocated by wider width nets, and more minimum width nets are allocated to their optimal gaps.

First, let $g_{\text{best}}(n, G)$ be the set of gaps such that the wirelength of net n is the minimum among the gaps in G when the trunk of the net $T(n)$ is allocated to the center of a gap, i.e.,

$$g_{\text{best}}(n, G) = \arg\min_{g \in G} l(n, y_{\text{mid}}(g)), \tag{5.1}$$

where $y_{\text{mid}}(g)$ is the y-coordinate of the center of gap g, i.e., $y_{\text{mid}}(g) = y(g) + \frac{w_g}{2}$.

Algorithm 4 Criticality-based CAP (CCAP)

Require: set of nets N_{in}, set of gaps G_{in}, width of net w_{target}

$N \leftarrow N_{\text{in}}$, $G \leftarrow G_{\text{in}}$
while $N \neq \emptyset$ **do** ▷ next gap
  $N_{\text{unit}} \leftarrow \{n \in N \mid w(n) = w_{\text{target}}\}$
  $g \leftarrow \text{CGO}(G, N_{\text{unit}})$
  delete(g, G)
  $N' \leftarrow \emptyset$, $C \leftarrow (w_{\text{g}})$
  $N \leftarrow \text{CriticalitySort}(N, g, G)$
  while $C \neq \emptyset$ **do** ▷ next round
    $Z \leftarrow \{x \mid d(x, N) = D(N)\}$
    $x \leftarrow -\infty$, $U \leftarrow N$, $c \leftarrow \text{top}(C)$
    while $U \neq \emptyset$ **do** ▷ next allocation
      $n \leftarrow \text{delete}(U)$
      if $x_{\min}(n) \leq x$ **then** ▷ violate HC
        continue
      end if
      if $(x, x_{\min}(n)) \cap Z \neq \emptyset$ **then** ▷ violate ZC
        continue
      end if
      $y \leftarrow \text{ceiling\_width}(n, N')$
      if $y + w(n) > c$ **then** ▷ violate CC
        continue
      end if
      $a(n) \leftarrow (g, y)$, $N' \leftarrow N' \cup \{n\}$ ▷ allocate n to g
      $x \leftarrow x_{\max}(n)$
      $C \leftarrow \text{insert}(y + w(n), C)$
      $N \leftarrow \text{delete}(n, N)$, $U \leftarrow N$
    end while
    if $x = -\infty$ **then** ▷ no allocation in round
      $C \leftarrow \text{delete}(c, C)$
    end if
  end while
end while

Gap congestion for gap g based on the set of nets N and gaps G is defined by

$$\text{GC}(g, G, N) = \sum_{n \in \{n' \in N \mid g \in g_{\text{best}}(n', G)\}} \frac{w(n)}{|g_{\text{best}}(n, G)|}. \tag{5.2}$$

The set of the least congested gaps based on the gap congestion is called *candidate gap*

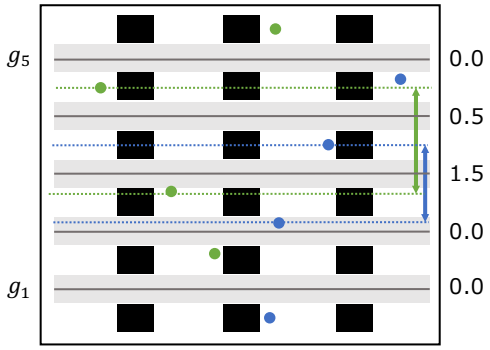


Figure 5.1: Gap congestion.

$g_{\text{candidate}}$ and is represented by

$$g_{\text{candidate}} = \arg\min_{g \in G} \text{GC}(g, G, N). \tag{5.3}$$

If $|g_{\text{candidate}}| > 1$, then algorithm selects an arbitrary gap in candidates as the target gap. In the thesis, the lowest positioned gap is selected as target gap, i.e.,

$$g_{\text{target}} = \arg\min_{g \in g_{\text{candidate}}} y_{\min}(g). \tag{5.4}$$

In CCAP, gap order is based on the gap congestion of the minimum wide nets, and then the set of candidate gaps is given by $g_{\text{candidate}} = \arg\min_{g \in G} \text{GC}(g, G, N_{\text{unit}})$, where N_{unit} is the set of the minimum width nets in the set of nets N. In the thesis, the minimum width is assumed to be 1, so $N_{\text{unit}} = \{n \in N \mid w(n) = 1\}$.

Fig. 5.1 shows an example of gap congestion with 2 nets which have 4 pins and 5 gaps $\{g_i\}_{i=1}^{5}$. The solid gray lines indicate the center of gaps. In the minimum wirelength, green net n_{green} can be allocated to the center of g_3 and g_4, and blue net n_{blue} can be allocated to the center of g_3. The candidate gaps are g_1, g_2, g_5, and the lowest positioned gap g_1 is selected as the target gap in CCAP.

5.2.2 Criticality-based Net Priority

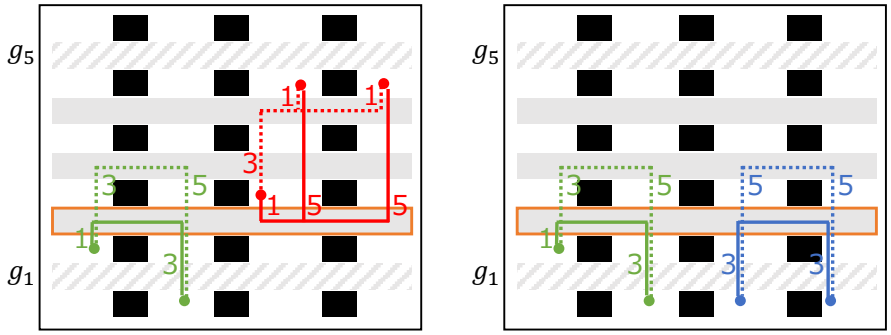
CCAP utilizes criticality-based net priority to allocate as many nets as possible in as small a wirelength as possible. The criticality is a measure of whether a net should be allocated to the target gap. The larger the criticality of a net, the larger the increase in wirelength when allocating the net is postponed. The criticality changes flexibly according to relative positions among a net, target gap, and remaining unused gaps.

The net priority of CCAP is defined based on three criteria:

(1) descending order of $w(n)$,

(2) descending order of criticality-based net priority,

(3) ascending order of $x_{\min}(n)$.



(a) $n_{\text{green}}(4), n_{\text{red}}(-6)$ **(b)** $n_{\text{green}}(4), n_{\text{blue}}(4)$

Figure 5.2: Examples of criticality-based net priority: gray striped boxes indicate gaps where nets have already been routed; gray boxes indicate remaining gaps and the gray box surrounded by orange lines indicates the target gap.

The two criteria, (1) and (3) are the same as CAP, and the newly introduced criterion (2) reduces the vertical wirelength.

The criticality is a difference of wirelength when horizontal trunk of a net $T(n)$ is allocated at the center of two gaps: the best gap g_{best} in given set of gaps and target gap g_{target}. Criticality for n with the two gaps, g_{best} and g_{target}, is formulated by

$$\text{Criticality}(n, g_{\text{best}}, g_{\text{target}}) = l(n, y_{\text{mid}}(g_{\text{best}})) - l(n, y_{\text{mid}}(g_{\text{target}})). \tag{5.5}$$

Note that g_{best} is an arbitrary element of $g_{\text{best}}(n, G)$ (Eq. (5.1)). Criticality-based net priority is defined by

$$\text{PC}(n, g_{\text{best}}, g_{\text{target}}, G) = \begin{cases} \text{Criticality}(n, g_{\text{best}}, g_{\text{target}}) & \text{if } \{G \setminus \{g_{\text{target}}\}\} \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

When there is no unused gap other than the target gap, all nets have the same priority.

Fig. 5.2 shows two examples of criticality-based net priority where the set of used gaps is $\{g_1, g_5\}$, $g_{\text{target}} = g_2$, and the set of remaining gaps is $\{g_3, g_4\}$. In the case of Fig. 5.2(a), if n_{green} is not allocated to the target gap g_2, the wirelength will increase. Thus, n_{green} has a high priority (Criticality$(n_{\text{green}}, g_3, g_2) = 4$). On the other hand, the wirelength of n_{red} allocated to the best gap g_4 is smaller than that allocated to the target gap g_2, and the criticality of n_{red} is smaller and has a lower priority (Criticality$(n_{\text{red}}, g_4, g_2) = -6$). When two nets have the same priority as shown in Fig. 5.2(b), they are prioritized according to the third criterion, the leftmost first order.

5.2.3 Time Complexity

Let m and k be the number of nets and the number of gaps, respectively. We assume that the number of pins of a net is bounded by a constant, and that k is $O(m)$.

The time complexity of the one-time execution of each step, except step 4 and step 7, in CCAP is $O(m)$. The time complexity of step 4 in which the target gap is selected is $O(km)$

Table 5.1: Setting (w indicates interval width).

scenario (prefix)	Pr(w) $w=1$	$w=2$	$w=3$	$w=4$
Sparse (c1)	0.80	0.10	0.08	0.02
Dense (c2)	0.50	0.30	0.15	0.05

since the gap congestion of each gap is obtained in $O(m)$. The time complexity of step 7 is $O(km + m\log m)$ since the criticality of each net is obtained in $O(k)$, and a sort of nets according to the priority is $O(m\log m)$.

In the following, the number of execution of each step in CCAP is discussed. Step 1 is executed once. N is set to N_{in}, and nets are removed one by one until N becomes empty. The numbers of execution of steps 2 to 7 are $O(m)$. For each N, the initial C is a subset of the set of the top boundaries of trunks allocated so far and the ceiling of a gap, and $|C|$ is $O(m)$. For each N, steps are executed at most once for each ceiling in the initial C for N. Therefore, the number of execution of the other steps is $O(m^2)$.

The total time complexity of steps 2 to 7 is $O(m)\times O(km + m\log m)$ which is $O(m^3)$, and the total time complexity of the other steps is $O(m^2)\times O(m)$ which is also $O(m^3)$. As the total, the time complexity of CCAP is $O(m^3)$.

5.2.4 Experimental Results

To confirm the validity of our CCAP, two experiments were conducted: algorithm comparison in wirelength used and the effect of gap order on wirelength in CCAP.

The first experiment compared three algorithms in the wirelength: Left-Edge [17]-based one (LE) mentioned in the Section 4.3, CAP [19], and CCAP. These algorithms are heuristics, and there is no guarantee that the number of gaps used is minimal. Thus, the chip size was set based on the number of gaps used the most among the compared algorithms, and LE used the most number of gaps. Note that the y-coordinates of pins are generated after the chip size is determined.

The second experiment used 7 gap orders for routing with the minimum number of gaps to measure the effect of gap order on the wirelength: random, bottom-up, top-down, and four variations of CGO. CGO is divided into four kinds: for gap congestion, use of nets of all width or of only minimum width, and routing from a congested gap and from an uncongested gap.

Benchmarks consist of the set of gaps and the set of nets. All gaps have the same vertical length, $w_{\text{g}} = 10$, and horizontal length, $l_{\text{c}} = 1$, and the separation between adjacent gaps is set to 10. The number of pins of each net is from 2 to 8 with equal probabilities, and each pin location is randomly generated as $x(p) \sim \mathcal{U}(0,1)$, $y(p) \sim \mathcal{U}(0,l_{\text{c}}), \forall p \in n$, where $w_{\text{c}} = |G_{\text{in}}| \times w_{\text{g}} + (|G_{\text{in}}|+1)\times 10$. The width of each trunk was selected among 1, 2, 3, and 4 according to the probabilities given in Table 5.1.

The algorithms were implemented by Python 3.10.9, and executed on Apple M2 CPU.

Table 5.2 shows comparison results in vertical wirelength. Note that there were gaps not used by CAP and CCAP since the chip size was adjusted to the number of gaps LE used. In the c1 benchmarks, CCAP reduced the wirelength by 10-45 points from the wirelength by CAP.

a Input Netlist and Lower Bound.

Netlist		Total	Density	Lower Bound		
name	#net	#pin		#gap	wirelength	
	$\|N_{\text{in}}\|$	$\|P_{\text{in}}\|$	$D(N_{\text{in}})$	$\lceil D(N_{\text{in}})/w_{\text{g}} \rceil$	$x/l_{\text{c}}/\|N_{\text{in}}\|$	$y/w_{\text{c}}/\|P_{\text{in}}\|$ (%)
c1-b1	10	51	14	2	0.67	0.2436 (100)
c1-b2	50	258	62	7	0.66	0.2103 (100)
c1-b3	100	519	118	12	0.61	0.2058 (100)
c1-b4	500	2,530	580	58	0.63	0.2101 (100)
c1-b5	1,000	4,924	1,151	116	0.63	0.2048 (100)
c2-b1	10	51	19	2	0.67	0.2436 (100)
c2-b2	50	258	81	9	0.66	0.2103 (100)
c2-b3	100	519	152	16	0.61	0.2058 (100)
c2-b4	500	2,530	777	78	0.63	0.2101 (100)
c2-b5	1,000	4,924	1,555	156	0.63	0.2048 (100)

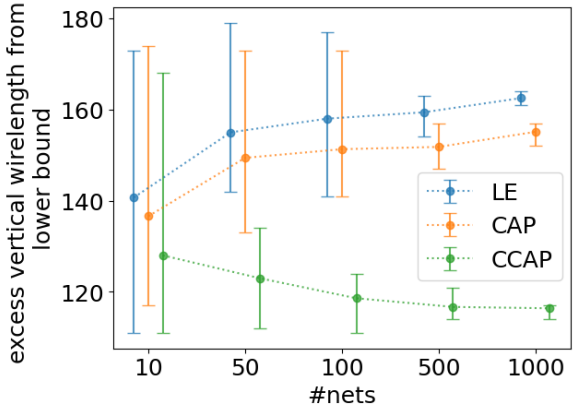
b Number of gaps used and wirelength.

Netlist		LE		CAP		CCAP (ours)	
name	#net	#gap	wirelength	#gap	wirelength	#gap	wirelength
c1-b1	10	2	0.3472 (143)	2	0.3275 (134)	2	0.2873 (118)
c1-b2	50	7	0.3214 (153)	7	0.2960 (141)	7	0.2763 (131)
c1-b3	100	12	0.3254 (158)	12	0.3231 (157)	12	0.2313 (112)
c1-b4	500	59	0.3227 (154)	58	0.3083 (147)	58	0.2415 (115)
c1-b5	1,000	117	0.3305 (161)	116	0.3204 (156)	116	0.2367 (116)
c2-b1	10	2	0.3416 (140)	2	0.3034 (125)	2	0.3034 (125)
c2-b2	50	9	0.3235 (154)	9	0.3194 (152)	9	0.2747 (131)
c2-b3	100	16	0.3277 (159)	16	0.3319 (161)	16	0.2575 (125)
c2-b4	500	79	0.3227 (154)	78	0.3313 (158)	78	0.2491 (119)
c2-b5	1,000	159	0.3315 (162)	156	0.3415 (167)	157	0.2389 (117)

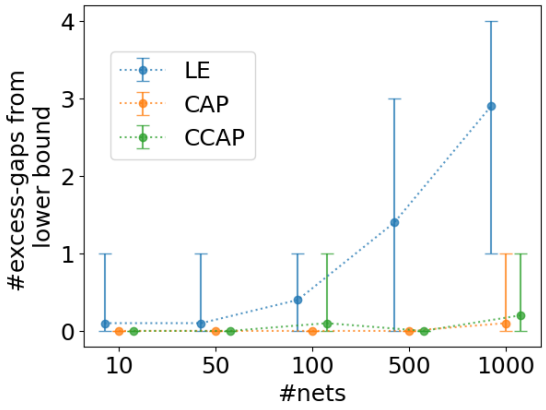
Table 5.2: Vertical wirelength.

In the c2 benchmarks with net counts up to 500, CCAP reduced the wirelength by 0-39 points compared to CAP. In the c2-b5 benchmark, only CAP used the minimum number of gaps, while CCAP needed to use one more than the number of gaps with a reduction in wirelength by 50 points. Introducing criticality-based net priority may increase the number of used gaps because it does not ensure that it accommodates nets more efficiently than CAP. The execution time of LE, CAP, and CCAP are 15.4, 353.9, and 303.8 seconds, respectively, for the c1-b5 benchmark.

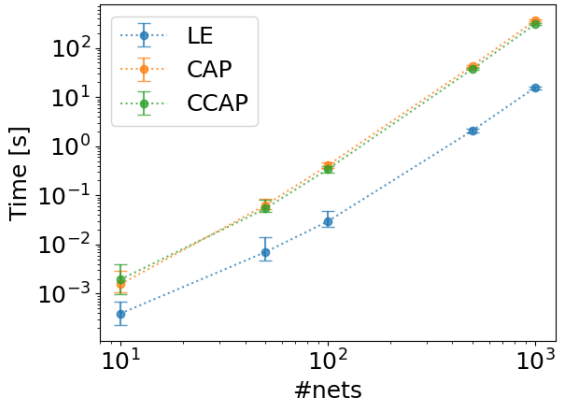
The results shown in Table 5.2 use only one benchmark for each condition. So, the variations of results were investigated. Fig. 5.3 shows the variation of allocation results with 10 c1 benchmarks including the benchmarks in Table 5.2. The 10 c1 benchmarks were generated with different random seeds. The points shown in each figure are the average value of corresponding metrics, and the error-bars represent the minimum and maximum values of the



(a) Excess vertical wirelength from lower bound.



(b) Number of excess gaps from lower bound.



(c) Computation time.

Figure 5.3: Variation in results with 10 different c1 benchmarks.

metrics.

Fig. 5.3(a) shows the variation of the excess vertical wirelength from the lower bound. As the number of nets increases, the variation of the results by each algorithm becomes smaller. For LE and CAP, the difference from the lower bound becomes larger as the number of nets increases, while for CCAP, the difference becomes smaller.

Fig. 5.3(b) shows the variation of the number of excess used gaps from the lower bound. For LE, the number of excess gaps increases as the number of nets increases, but CAP and CCAP do not increase significantly. Comparing the maximum number of excess gaps for the 10 benchmarks, CAP and CCAP are at most 1. Their allocations use gaps closer to the lower bound.

Fig. 5.3(c) shows the computation time of the algorithms. In the case of the small number of nets, the variation is larger, but there is almost no difference in the case of the large number of nets. The time complexities of LE, CAP, and CCAP, estimated by the differences of the average computation times between $|N_{\text{in}}| = 100$ and 1,000, are $O\left(m^{2.31}\right)$, $O\left(m^{2.59}\right)$, and $O\left(m^{2.67}\right)$, respectively.

Table 5.3 shows the effect of gap order on the wirelength by using the same benchmarks used in Table 5.2. Note that GCF and GCA stand for gap congestion first that prioritizes congested

gaps and gap congestion avoidance that prioritizes uncongested gaps, respectively. In the c1 benchmarks, when the number of nets was 50 or less, algorithms that achieved the smallest wirelength vary depending on benchmarks, and otherwise, the wirelength of GCA was the smallest. Since the main target of CCAP is a large benchmark, GCA is suitable for more than 100 nets. The use of wide nets or not for gap congestion does not have a significant effect on the wirelength since c1 benchmarks are mostly a single-width net.

In all c2 benchmarks except for the c2-b5, algorithms that achieved the smallest wirelength depends on the benchmark, and the wirelength by GCA is relatively smaller than ones by other gap orders in many cases. In the case of gap congestion using only the minimum width nets, the amount of improvement in wirelength depends on the positions of the wide nets. Depending on the input nets, the nets used for the gap congestion can be varied to reduce the wirelength more. In the c2-b5 benchmark, only some of the orders completed the routing with the minimum number of gaps, and random order achieved the shortest wirelength among them. Like c2-b5 benchmark, when the proposed method cannot allocate some nets in a limited number of gaps, it may be possible to allocate them while reducing the wirelength by simply changing the allocation order of gaps.

a Simple gap order.

Netlist name	#net	#gap	Random	Bottom-up	Top-down
c1-b1	10	2	0.2873 (118)	0.2873 (118)	0.2811 (115)
c1-b2	50	7	0.2412 (115)	0.2420 (115)	0.2521 (120)
c1-b3	100	12	0.2590 (126)	0.2721 (132)	0.2608 (127)
c1-b4	500	58	0.2468 (117)	0.2638 (126)	0.2702 (129)
c1-b5	1,000	116	0.2363 (115)	0.2665 (130)	0.2608 (127)
c2-b1	10	2	0.3034 (125)	0.3034 (125)	0.3276 (134)
c2-b2	50	9	0.2961 (141)	0.2677 (127)	0.2751 (131)
c2-b3	100	16	0.2690 (131)	0.2939 (143)	0.2761 (134)
c2-b4	500	78	0.2467 (117)	0.2989 (142)	0.3000 (143)
c2-b5	1,000	156	0.2456 (120) [a]	0.3027 (148)	N/A

[a] Four of the five runs allocate all nets to the minimum gaps.

b Various gap congestion-based orders.

Netlist name	#net	#gap	GCF with N_{in}	GCA with N_{in}	GCF with N_{unit}	GCA with N_{unit}
c1-b1	10	2	0.2811 (115)	0.2873 (118)	0.2811 (115)	0.2873 (118)
c1-b2	50	7	0.2503 (119)	0.2763 (131)	0.2503 (119)	0.2763 (131)
c1-b3	100	12	0.2552 (124)	0.2330 (113)	0.2571 (125)	0.2313 (112)
c1-b4	500	58	0.2600 (124)	0.2387 (114)	0.2597 (124)	0.2387 (114)
c1-b5	1,000	116	0.2556 (125)	0.2334 (114)	0.2561 (125)	0.2337 (114)
c2-b1	10	2	0.3276 (134)	0.3034 (125)	0.3276 (134)	0.3034 (125)
c2-b2	50	9	0.2541 (121)	0.2732 (130)	0.2919 (139)	0.2747 (131)
c2-b3	100	16	0.2924 (142)	0.2619 (127)	0.2917 (142)	0.2575 (125)
c2-b4	500	78	0.2789 (133)	0.2423 (115)	0.2759 (131)	0.2493 (119)
c2-b5	1,000	156	0.2776 (136)	N/A	0.2747 (134)	N/A

Table 5.3: Vertical wirelength.

5.3 Gap Swap-Flip Algorithm (GSF)

The previous section attempts to minimize the wirelength by the heuristic method. However, there is still room for further wirelength reduction, and this section proposes a post-processing method to reduce the wirelength, called *gap swap-flip* (GSF). Given an initial allocation, GSF first reduces the wirelength outside gaps by swapping the allocation between two gaps, called *gap swap* (GS), and then reduces the wirelength inside a gap by reversing the allocation within the gap, called *gap flip* (GF). Each step iteratively changes allocation in greedy manner until no more improvement can be made.

To explain each step of GSF in detail, let this section introduce some definitions. In GCR, the allocation of the trunk of a net is represented by (g, s) where g is the gap allocated and s is the offset of the trunk in the allocated gap. When net n is allocated to (g, s), the y-coordinate of the trunk of the net is given by $y(n, g, s) = y(g) + s$. The vertical wirelength of net n allocated to (g, s) is defined as $l(n, g, s) = \sum_{p \in n} |y(n, g, s) - y(p)|$.

The first step of GSF, GS, swaps the allocation of nets between two gaps to reduce the wirelength outside gaps. The set of nets that have allocated to gap g by allocation $\mathcal{A}$ is given by $N(g, \mathcal{A}) = \{n \in N_{\text{in}} \mid \mathcal{G}(n) = g\}$. The allocation $\mathcal{A}'$ obtained from allocation $\mathcal{A}$ by GS for two gaps g, g' is defined as

$$\mathcal{A}'(n) = \begin{cases} (g', \mathcal{S}(n)) & \text{if } n \in N(g, \mathcal{A}), \\ (g, \mathcal{S}(n)) & \text{if } n \in N(g', \mathcal{A}), \\ \mathcal{A}(n) & \text{otherwise.} \end{cases}$$

The difference of GS in the wirelength is

$$\mathrm{D}_{\text{swap}}(g, g') = \mathrm{D}_{\text{m}}(g, g') + \mathrm{D}_{\text{m}}(g', g), \tag{5.6}$$

where

$$\mathrm{D}_{\text{m}}(g, g') = \sum_{n \in N(g, \mathcal{A})} l(n, g', \mathcal{S}(n)) - \sum_{n \in N(g, \mathcal{A})} l(n, g, \mathcal{S}(n)). \tag{5.7}$$

GS swaps the allocation if and only if $\mathrm{D}_{\text{swap}}(g, g') < 0$.

The second step of GSF, GF, flips the allocation of nets in a gap vertically, i.e., the allocation $\mathcal{A}'$ obtained from $\mathcal{A}$ by GF for gap g is defined as

$$\mathcal{A}'(n) = \begin{cases} (g, w_{\text{g}} - \mathcal{S}(n)) & \text{if } n \in N(g, \mathcal{A}), \\ \mathcal{A}(n) & \text{otherwise.} \end{cases}$$

GF is used to reduce the wirelength inside the gap. The difference of GF in wirelength is

$$\mathrm{D}_{\text{flip}}(g) = \sum_{n \in N(g, \mathcal{A})} l\left(n, g, w_{\text{g}} - \mathcal{S}(n)\right) - \sum_{n \in N(g, \mathcal{A})} l(n, g, \mathcal{S}(n)). \tag{5.8}$$

GF is performed if and only if $\mathrm{D}_{\text{flip}}(g) < 0$.

5.3.1 Experimental Results

To evaluate GSF, routing solutions obtained by Left-Edge [17]-based one (LE) mentioned in the Section 4.3, CAP [19], and CCAP [20] are used as initial solutions and are applied to 3 variations of GSF: GS, GF, and GSF.

The benchmarks are the same as those used in the previous section (Table 5.2 and Table 5.3). Algorithms were implemented by Python 3.10.9, and executed on Apple M2 CPU. Given LE solution for the largest benchmark, the execution time of GS and GF was up to 78 seconds and 10 microseconds, respectively.

Table 5.4 and Table 5.5 show the results in vertical wirelength with c1 and c2 benchmarks, respectively. Note that there were gaps not used by CAP and CCAP since the chip size was adjusted to the number of gaps LE uses. It is confirmed that the wirelength is reduced from the initial results of all algorithms. When the number of nets is more than 100, the LE and CAP algorithms, which do not consider the wirelength, do not fall below 130 even with GSF. On the other hand, CCAP is below 130 from the initial solution, and GSF further reduces the wirelength.

5.4 Summary

The wirelength reduction is critical in improving chip performance, as it influences power consumption, congestion, and so on. In this chapter, two algorithms, CCAP and GSF, are proposed for GCR to minimize wirelength.

CCAP is conscious of congested gaps and determines the order of gaps and nets so that as many nets as possible are routed with the smaller wirelength as much as possible. Experimental results show that CCAP significantly reduced the wirelength while using almost the same number of gaps as a lower bound.

GSF reduces the wirelength inside and outside gaps in two steps and contributes to obtaining a high-quality chip efficiently. Experimental results show that GSF reduces the wirelength of the initial solutions by three algorithms, LE, CAP, and CCAP.

By using these proposed methods, high-performance chip designs can be obtained efficiently.

a Input Netlist and Lower Bound

Netlist		Total	Density	Lower Bound	
name	#net	#pin		#gap	wirelength
	$\|N_{in}\|$	$\|P_{in}\|$	$D(N_{in})$	$\lceil D(N_{in})/w_g \rceil$	$x/l_c/\|N_{in}\|$ $y/w_c/\|P_{in}\|$ (%)
c1-b1	10	51	14	2	0.67 0.2436 (100)
c1-b2	50	258	62	7	0.66 0.2103 (100)
c1-b3	100	519	118	12	0.61 0.2058 (100)
c1-b4	500	2,530	580	58	0.63 0.2101 (100)
c1-b5	1,000	4,924	1,151	116	0.63 0.2048 (100)

b Left-Edge (LE)

Netlist		Used	Vertical wirelength $y/w_c/\|P_{in}\|$ (%)			
name	#net	#gap	LE	LE + GS	LE + GF	LE + GSF (ours)
c1-b1	10	2	0.3472 (143)	0.2900 (119)	0.3404 (140)	0.2766 (114)
c1-b2	50	7	0.3214 (153)	0.2557 (122)	0.3194 (152)	0.2541 (121)
c1-b3	100	12	0.3254 (158)	0.2833 (138)	0.3244 (158)	0.2821 (137)
c1-b4	500	59	0.3227 (154)	0.2908 (138)	0.3224 (153)	0.2904 (138)
c1-b5	1,000	117	0.3305 (161)	0.2877 (140)	0.3303 (161)	0.2875 (140)

c CAP

Netlist		Used	Vertical wirelength $y/w_c/\|P_{in}\|$ (%)			
name	#net	#gap	CAP	CAP + GS	CAP + GF	CAP + GSF (ours)
c1-b1	10	2	0.3275 (134)	0.2886 (118)	0.3275 (134)	0.2766 (114)
c1-b2	50	7	0.2960 (141)	0.2593 (123)	0.2931 (139)	0.2560 (122)
c1-b3	100	12	0.3231 (157)	0.2747 (133)	0.3223 (157)	0.2738 (133)
c1-b4	500	58	0.3083 (147)	0.2829 (135)	0.3081 (147)	0.2825 (134)
c1-b5	1,000	116	0.3204 (156)	0.2845 (139)	0.3203 (156)	0.2843 (139)

d CCAP

Netlist		Used	Vertical wirelength $y/w_c/\|P_{in}\|$ (%)			
name	#net	#gap	CCAP	CCAP + GS	CCAP + GF	CCAP + GSF (ours)
c1-b1	10	2	0.2873 (118)	0.2873 (118)	0.2838 (116)	0.2838 (116)
c1-b2	50	7	0.2763 (131)	0.2487 (118)	0.2747 (131)	0.2486 (118)
c1-b3	100	12	0.2313 (112)	0.2298 (112)	0.2305 (112)	0.2292 (111)
c1-b4	500	58	0.2415 (115)	0.2371 (113)	0.2413 (115)	0.2369 (113)
c1-b5	1,000	116	0.2367 (116)	0.2292 (112)	0.2366 (116)	0.2290 (112)

Table 5.4: Vertical wirelength with c1 benchmarks.

a Input Netlist and Lower Bound

Netlist		Total	Density	Lower Bound		
name	#net	#pin		#gap	wirelength	
	$\|N_{\text{in}}\|$	$\|P_{\text{in}}\|$	$D(N_{\text{in}})$	$\lceil D(N_{\text{in}})/w_{\text{g}} \rceil$	$x/l_{\text{c}}/\|N_{\text{in}}\|$	$y/w_{\text{c}}/\|P_{\text{in}}\|$ (%)
c2-b1	10	51	19	2	0.67	0.2436 (100)
c2-b2	50	258	81	9	0.66	0.2103 (100)
c2-b3	100	519	152	16	0.61	0.2058 (100)
c2-b4	500	2,530	777	78	0.63	0.2101 (100)
c2-b5	1,000	4,924	1,555	156	0.63	0.2048 (100)

b Left-Edge (LE)

Netlist		Used	Vertical wirelength $y/w_{\text{c}}/\|P_{\text{in}}\|$ (%)			
name	#net	#gap	LE	LE + GS	LE + GF	LE + GSF (ours)
c2-b1	10	2	0.3416 (140)	0.3120 (128)	0.3208 (132)	0.2956 (121)
c2-b2	50	9	0.3235 (154)	0.2634 (125)	0.3217 (153)	0.2609 (124)
c2-b3	100	16	0.3277 (159)	0.2864 (139)	0.3263 (159)	0.2854 (139)
c2-b4	500	79	0.3227 (154)	0.2873 (137)	0.3224 (153)	0.2871 (137)
c2-b5	1,000	159	0.3315 (162)	0.2820 (138)	0.3314 (162)	0.2818 (138)

c CAP

Netlist		Used	Vertical wirelength $y/w_{\text{c}}/\|P_{\text{in}}\|$ (%)			
name	#net	#gap	CAP	CAP + GS	CAP + GF	CAP + GSF (ours)
c2-b1	10	2	0.3034 (125)	0.3034 (125)	0.2979 (122)	0.2979 (122)
c2-b2	50	9	0.3194 (152)	0.2648 (126)	0.3181 (151)	0.2620 (125)
c2-b3	100	16	0.3319 (161)	0.2693 (131)	0.3314 (161)	0.2679 (130)
c2-b4	500	78	0.3313 (158)	0.2809 (134)	0.3312 (158)	0.2807 (134)
c2-b5	1,000	156	0.3415 (167)	0.2776 (136)	0.3414 (167)	0.2775 (135)

d CCAP

Netlist		Used	Vertical wirelength $y/w_{\text{c}}/\|P_{\text{in}}\|$ (%)			
name	#net	#gap	CCAP	CCAP + GS	CCAP + GF	CCAP + GSF (ours)
c2-b1	10	2	0.3034 (125)	0.3034 (125)	0.2979 (122)	0.2979 (122)
c2-b2	50	9	0.274 (131)	0.2544 (121)	0.2724 (130)	0.2506 (119)
c2-b3	100	16	0.2575 (125)	0.2437 (118)	0.2559 (124)	0.2416 (117)
c2-b4	500	78	0.2491 (119)	0.2387 (114)	0.2489 (118)	0.2385 (114)
c2-b5	1,000	157	0.2389 (117)	0.2306 (113)	0.2388 (117)	0.2305 (113)

Table 5.5: Vertical wirelength with c2 benchmarks.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The dissertation dealt with the routing problem for advanced 3D chip design by hybrid bonding.

First, the routing problem of the 3D chip design was modeled as gap channel routing, where it is necessary to accomplish routing various-width trunks of nets into limited multiple gaps.

Second, the dissertation proposed CAP accomplishing routing with the minimum number of gaps possible. CAP uses fewer gaps compared with conventional algorithms by prioritizing wide trunks of nets and filling in the gaps as much as possible. By using as few gaps as possible, the free gaps can be used for other optimizations like wirelength reduction, contributing to the realization of higher-performance chip designs.

Third, the dissertation proposed CCAP, which extends CAP to reduce the wirelength while minimizing the increase in the number of gaps used. CCAP is conscious of congested gaps and determines the order of gaps and nets so that as many nets as possible are allocated with the smaller wirelength as much as possible. Compared with conventional algorithms, CCAP significantly reduces the wirelength while using almost the same number of gaps as a lower bound. Such a small total wirelength contributes to the design closure.

Finally, the dissertation proposed GSF that minimizes the wirelength from the given initial allocation. Without changing the number of gaps used, the wirelength is further reduced from the initial solution.

These routing algorithms proposed in the dissertation contribute to the development of high-performance advanced 3D chip design.

6.2 Future Work

This dissertation focuses on routing algorithms that minimize routing area and wirelength and does not aim at proposing any others such as detailed routing required in the later of the design process. It is necessary to extend the scope of the study to the algorithms required for subsequent chip design. For example, this dissertation did not consider routing feasibility perpendicular to the bottleneck direction, but in reality, there may be obstacles even on the vertical routing layer. Therefore, the routing is not completed unless the bottleneck direction routing is done after considering obstacles in other layers. In addition, it is necessary to optimize not

only the routing area and wirelength of routing but also various indicators for high performance like the signal delay difference among nets. The realization of a detailed routing method that performs routing while optimizing these parameters will contribute to the development of 3D chips with higher performance.

References

[1] G. E. Moore, “Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff.,” *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 33–35, 2006.

[2] L. Clavelier, C. Deguet, L. Di Cioccio, E. Augendre, A. Brugere, P. Gueguen, Y. Le Tiec, H. Moriceau, M. Rabarot, T. Signamarcheix, J. Widiez, O. Faynot, F. Andrieu, O. Weber, C. Le Royer, P. Batude, L. Hutin, J.-F. Damlencourt, S. Deleonibus, and E. Defaÿ, “Engineered substrates for future more moore and more than moore integrated devices,” in *2010 International Electron Devices Meeting*, pp. 2.6.1–2.6.4, 2010.

[3] S. Zhang, Z. Li, H. Zhou, R. Li, S. Wang, K.-W. Paik, and P. He, “Challenges and recent prospectives of 3d heterogeneous integration,” *e-Prime - Advances in Electrical Engineering, Electronics and Energy*, vol. 2, p. 100052, 2022.

[4] P. Ramm, J. J.-Q. Lu, and M. M. Taklo, *Handbook of wafer bonding*. John Wiley & Sons, 2012.

[5] S.-A. Chew, B. Zhang, K. Vanstreels, E. Chery, J. De Messemaeker, L. Witters, K. Van Sever, S. Iacovo, S. Dewilde, M. Stucchi, J. De Vos, G. Beyer, A. Miller, and E. Beyne, “The challenges and solutions of cu/sicn wafer-to-wafer hybrid bonding scaling down to 400nm pitch,” in *2023 International Electron Devices Meeting (IEDM)*, pp. 1–4, 2023.

[6] P. Shankara K., *Advanced Wirebond Interconnection Technology*. Springer, 2004.

[7] H. George G., *Wire Bonding in Microelectronics*. McGraw-Hill, 2010.

[8] D. B. Ingerly, S. Amin, L. Aryasomayajula, A. Balankutty, D. Borst, A. Chandra, K. Cheemalapati, C. S. Cook, R. Criss, K. Enamul, W. Gomes, D. Jones, K. C. Kolluru, A. Kandas, G.-S. Kim, H. Ma, D. Pantuso, C. Petersburg, M. Phen-givoni, A. M. Pillai, A. Sairam, P. Shekhar, P. Sinha, P. Stover, A. Telang, and Z. Zell, “Foveros: 3D integration and the use of face-to-face chip stacking for logic devices,” in *2019 IEEE International Electron Devices Meeting (IEDM)*, pp. 19.6.1–19.6.4, 2019.

[9] J. H. Lau, “Recent advances and trends in cu–cu hybrid bonding,” *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 13, no. 3, pp. 399–425, 2023.

[10] J. H. Lau, "State of the art of cu–cu hybrid bonding," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 14, no. 3, pp. 376–396, 2024.

[11] J. Park, B. Lee, H. Lee, D. Lim, J. Kang, C. Cho, M. Na, and I. Jin, "Wafer to wafer hybrid bonding for DRAM applications," in *2022 IEEE 72nd Electronic Components and Technology Conference (ECTC)*, pp. 126–129, 2022.

[12] Y. Kagawa, N. Fujii, K. Aoyagi, Y. Kobayashi, S. Nishi, N. Todaka, S. Takeshita, J. C. Taura, H. Takahashi, Y. Nishimura, K. Tatani, M. Kawamura, H. Nakayama, T. Nagano, K. Ohno, H. Iwamoto, S. Kadomura, and T. Hirayama, "Novel stacked cmos image sensor with advanced cu2cu hybrid bonding," *2016 IEEE International Electron Devices Meeting (IEDM)*, pp. 8.4.1–8.4.4, 2016.

[13] M. Tagami, "Cmos directly bonded to array (cba) technology for future 3d flash memory," in *Proc. International Electron Devices Meeting (IEDM)*, 2023.

[14] S. Kobayashi, K. Tashiro, Y. Minemura, K. Nakagami, K. Arita, T. Oohashi, K. Funayama, H. Sakai, M. Mushiga, K. Okabe, Y. Kanno, S. Shimizu, E. Fujikura, A. Nakae, K. Yamaguchi, H. Yamawaki, K. Nakajima, and M. Sato, "High peformance 3d flash memory with 3.2gbps interface and 205mb/s program throughput based on cba(cmos directly bonded to array) technology," in *Proc. International Electron Devices Meeting (IEDM)*, 2023.

[15] Y. Ouyang, S. Yang, D. Yin, X. Huang, Z. Wang, S. Yang, K. Han, and Z. Xia, "Excellent reliability of xtacking™ bonding interface," in *2021 IEEE International Reliability Physics Symposium (IRPS)*, pp. 1–6, 2021.

[16] R. K. Pal, *Multi-Layer Channel Routing: Complexity and Algorithms*. Narosa Publishing House, 2000.

[17] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," in *Proc. 8th Design Automation Workshop (DAC)*, pp. 155–169, 1971.

[18] B. Krishna, C. Chen, and N. Sehgal, "Routing wires with non-uniform width and spacing in data paths," in *Proc. 11th International Conference on Microelectronics (ICM)*, pp. 85–88, 1999.

[19] M. Shimoda and A. Takahashi, "Gridless gap channel routing with variable-width wires," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 108, no. 3, 2025. (in press).

[20] M. Shimoda and A. Takahashi, "Generalized gap channel routing to minimize wirelength," *IPSJ Trans. on System LSI Design Methodology*, vol. 18, 2025. (submitted).

[21] M. Shimoda and A. Takahashi, "Wirelength minimization by gap swap-flip in gridless gap channel routing," in *Proc. IEEE International SoC Design Conference (ISOCC)*, 2024.

[22] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, *Detailed Routing*, pp. 171–194. Cham: Springer International Publishing, 2022.

[23] T. Yoshimura and E. Kuh, "Efficient algorithms for channel routing," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 1, no. 1, pp. 25–35, 1982.

[24] D. N. Deutsch, "A dogleg channel router," *DAC '76: Proceedings of the 13th Design Automation Conference*, p. 425–433, 1976.

[25] T.-T. Ho, S. Iyengar, and S.-Q. Zheng, "A general greedy channel routing algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 2, pp. 204–211, 1991.

[26] S. S. Sau, A. Pal, T. N. Mandal, A. K. Datta, R. K. Pal, and A. Chaudhuri, "A graph based algorithm to minimize total wire length in VLSI channel routing," in *2011 IEEE International Conference on Computer Science and Automation Engineering*, vol. 3, pp. 61–65, 2011.

[27] D. Hightower and R. Boyd, "A generalized channel router," in *Proc. 17th Design Automation Conference (DAC)*, pp. 12–21, 1980.

[28] K. Sato, H. Shimoyama, T. Nagai, M. Ozaki, and T. Yahara, "A "grid-free" channel router," in *Proc. 17th Design Automation Conference (DAC)*, pp. 22–31, 1980.

[29] H. Chen and E. Kuh, "Glitter: A gridless variable-width channel router," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 5, no. 4, pp. 459–465, 1986.

[30] C. H. Ng, "A "gridless" variable-width channel router for marco cell design," in *Proc. 24th Design Automation Conference (DAC)*, pp. 633–636, 1987.

[31] P. Groeneveld, "A multiple layer contour-based gridless channel router," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 12, pp. 1278–1288, 1990.

[32] D. B. Polkl, "A three-layer gridless channel router with compaction," in *Proc. 24th Design Automation Conference (DAC)*, pp. 146–151, 1987.

[33] H.-J. Rothermel and D. Mlynski, "Automatic variable-width routing for VLSI," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 2, no. 4, pp. 271–284, 1983.

[34] K. Taniguchi, S. Tayu, A. Takahashi, M. Molongo, M. Minami, and K. Nishioka, "Two-layer bottleneck channel track assignment for analog VLSI," *IPSJ Trans. on System LSI Design Methodology*, vol. 17, 2024. (to appear).

[35] Z. Wang, M. Shimoda, and A. Takahashi, "BCA channel routing to minimize wirelength for generalized channel problem," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, 2024.

[36] M. Guruswamy and D. Wong, "Echelon: a multilayer detailed area router," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1126–1136, 1996.

[37] C. Chu and Y.-C. Wong, "Flute: Fast lookup table based rectilinear steiner minimal tree algorithm for vlsi design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 70–83, 2008.

[38] L. McMurchie and C. Ebeling, "Pathfinder: A negotiation-based performance-driven router for fpgas," in *Third International ACM Symposium on Field-Programmable Gate Arrays*, pp. 111–117, 1995.

[39] J. He, M. Burtscher, R. Manohar, and K. Pingali, "Sproute: A scalable parallel negotiation-based global router," in *Proceedings of 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2019.

[40] M. Kim, H. Park, S. Kim, K. Son, S. Kim, K. Son, S. Choi, G. Park, and J. Kim, "Reinforcement learning-based auto-router considering signal integrity," in *2020 IEEE 29th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*, pp. 1–3, 2020.

[41] L. Li, Y. Cai, and Q. Zhou, "A survey on machine learning-based routing for vlsi physical design," *Integration*, vol. 86, pp. 51–56, 2022.

[42] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*. New York: Freeman and Co., 1979.

[43] E. G. Coffman, J. Csirik, G. Galambos, S. Martello, D. Vigo, *et al.*, "Bin packing approximation algorithms: Survey and classification.," in *Handbook of combinatorial optimization*, pp. 455–531, Springer, 2013.

[44] T. Yokomaru, T. Izumi, A. Takahashi, and Y. Kajitani, "Solution of integer bin packing problem with fixed capacity by FFD," *IPSJ SIG Technical Reports (95-DA-76)*, vol. 95, no. 72, pp. 1–8, 1995. (in Japanese).

[45] T. Yokomaru, T. Izumi, and Y. Kajitani, "The FFD bin-packing algorithm and its extension for VLSI circuit partitioning," *IEICE Trans. Fundamentals*, vol. J80-A, no. 9, pp. 1452–1459, 1997. (in Japanese).

Publications

The work in this dissertation is based on the following publications:

Peer-reviewed Journal Papers

- M. Shimoda and A. Takahashi, “Gridless Gap Channel Routing with Variable-width Wires,” IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences, Vol. 108, No. 3 (2025).
- M. Shimoda and A. Takahashi, “Gridless Gap Channel Routing to Minimize Wirelength,” IPSJ Trans. on System LSI Design Methodology, Vol. 18 (2025).

Peer-reviewed Conference Papers

- M. Shimoda and A. Takahashi, “Wirelength Minimization by Gap Swap-Flip in Gridless Gap Channel Routing,” Proc. IEEE International SoC Design Conference (ISOCC) (2024).