論文 / 著書情報
Article / Book Information

論題(和文)	
Title(English)	Gridless Gap Channel Routing to Minimize Wirelength
著者(和文)	下田 将之, 高橋 篤司
Authors(English)	Masayuki Shimoda, Atsushi Takahashi
出典(和文)	, Vol. 18, , pp. 2-9
Citation(English)	IPSJ Transactions on System and LSI Design Methodology, Vol. 18, , pp. 2-9
発行日 / Pub. date	2025, 2
権利情報 / Copyright	

IPSJ Transactions on System and LSI Design Methodology Vol.18 2–9 (Feb. 2025)

[DOI: 10.2197/ipsjtsldm.18.2]

Regular Paper

Gridless Gap Channel Routing to Minimize Wirelength

Masayuki Shimoda[1,a)] Atsushi Takahashi[1,b)]

Received: June 5, 2024, Accepted: October 31, 2024

Abstract: Gridless Gap Channel Routing (GGCR) is a routing problem defined in a critical routing layer in an advanced chip where a single horizontal trunk Steiner tree connects each net. The trunk widths are not unique, and trunks are allocated to partitioned routing areas, called gaps, without overlap. This paper proposes criticality-based ceiling and packing (CCAP) for GGCR to shorten the wirelength. CCAP routes a net with a smaller wirelength as much as possible by considering the congestion of gaps and the criticality of nets in terms of wirelength.

Keywords: channel routing, gap-channel, gridless, single trunk steiner tree

1. Introduction

With recent technological advances, some types of advanced chips contain several components such as sensors, liquid crystal displays (LCDs), controllers, and bonding pads for three-dimensional (3D) stacking integration such as complementary metal-oxide-semiconductor (CMOS) directly bonded to array (CBA) technology in 3D flash memory [5], [13], which are scattered as regular patterns in routing layers and treated as obstacles during the routing phase. In some routing layers in such advanced chips [16], gaps between obstacles and a gap channel are defined. It tends to be a routing bottleneck since many nets need to use them for their connections [14], [15]. It is necessary to use them efficiently to accomplish routing with limited gaps. Pins of nets are placed even inside the routing area since various modules are scattered throughout the chip in addition to obstacles. The variety of wire widths of nets needed inhibits the efficient use of gaps. A gap channel is treated as gridless, and it is preferred to complete routing using a single horizontal wire segment for each net, i.e., dogleg is not preferred.

Figure 1 shows an example of routing layers in such chips, called critical routing layers in this paper, where a gap channel is defined. It is a top view of the chip, and modules as well as vertical wires not in critical routing layers are drawn in the detailed view in the right. The areas horizontally traversing between the obstacles are routing areas (gray rectangles) called *gaps*. A gridless gap channel is defined as the union of these gaps.

The routing problem mentioned above, mainly focusing on horizontal wire segments of nets, is modeled as Generalized Channel Routing (GCR) for unit width wires in Ref. [16], and is modeled as Gridless Gap Channel Routing (GGCR) for various width wires in Ref. [12]. GCR and GGCR are defined to model a situation where a feasibility check is required in the early design stage. Due to a lack of horizontal routing resources in GCR and GGCR, it is expected that single horizontal wire segments of given nets, called trunks, are allocated to given routing areas, and that the detailed routing for the vertical wires is not taken into account. In case where conflicts among vertical wires occur in a followed design stage, it is expected that they are resolved by utilizing flexibility in module design and location in followed detailed placement and routing. No matter how many critical layers there are, algorithms for GCR and GGCR will be adapted to each of the critical layers.

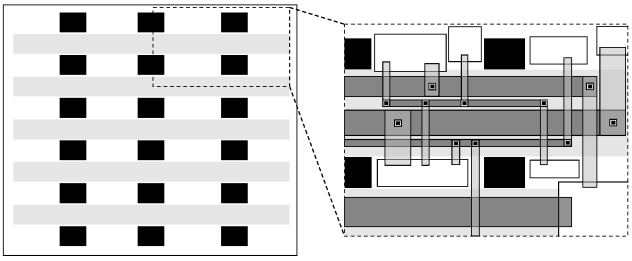


Fig. 1 Gridless gap channel [12]. Gaps (gray) are defined horizontally between obstacles (black) and are regularly arranged. The pins of nets are defined on modules (while rectangles) inside the area. The width of horizontal wires (dark gray) of nets varies.

In GGCR, the channel consists of vertically stacked fixed-width gaps in a routing layer when viewed from the top, and each net is connected by using the trunk of the net with a specified wire width. The trunk of a net is allocated to a gap in the channel so that no overlaps with other trunks occur. The objective of GGCR in this paper is to allocate trunks of nets in given gaps while achieving a small wirelength. A small total wirelength contributes to the design closure. In GGCR, it is achieved by the reduction of vertical segments. The total wirelength of vertical segments of a net that connect between the pins of the net and the horizontal trunk of the net depends on the y-coordinate at which the trunk is allocated. The location of the horizontal trunk is limited to the inside of gaps, and the wirelength is greatly affected by which gap is used.

Most conventional channel routing algorithms, including Left-Edge (LE) algorithm [3], which is popular in conventional chan-

[1] Tokyo Institute of Technology, Meguro, Tokyo 152–8550, Japan
[a)] shimoda@eda.ict.e.titech.ac.jp
[b)] atsushi@ict.e.titech.ac.jp

nel routing, neither consider the divided routing area (gaps) nor take the widths of wires into account and cannot obtain reasonable solutions in GGCR. Therefore, a dedicated routing algorithm is desired, especially for GGCR. As for the efficient method available for GGCR, Ceiling-and-Packing Algorithm (CAP) was proposed [12]. CAP uses gaps efficiently by prioritizing wide trunks of nets and filling gaps as much as possible with ceiling and zone constraints. However, CAP is intended only to minimize the number of gaps, not to reduce the wirelength.

This paper proposes criticality-based CAP (CCAP) to reduce the wirelength in GGCR. CCAP is based on CAP and adopts congestion-aware gap order (CGO) and criticality-based net priority. CCAP routes wide nets first to fill gaps as much as possible as CAP does, but these prioritized nets may prevent achieving a smaller wirelength of remaining nets. CGO prioritizes gaps that are unlikely to be used to achieve small wirelength of remaining nets and helps to achieve a small wirelength of remaining nets. Additionally, criticality-based net priority prioritizes nets that need additional wirelength when routed to a gap processed later and helps to avoid an increase in the wirelength of remaining nets. Incorporating them into CAP reduces the wirelength significantly while completing the routing by using gaps required for CAP in most cases.

The rest of the paper is organized as follows: Section 2 and Section 3 explain GGCR and its related works. Section 4 explains CAP, and Section 5 describes the proposed CCAP. Section 6 shows the experimental results comparing CCAP and conventional algorithms, and Section 7 presents the conclusion.

2. Wirelength Minimization in GGCR

In GGCR, a critical routing layer is modeled as a *gridless gap channel*, and the horizontal trunk of a net is allocated to a gap in the channel. The channel is a rectangle routing area whose vertical and horizontal lengths are C_v and C_h, respectively, and the coordinate of the channel is defined so that the lower left corner of the channel is $(0, 0)$. The channel has a set of gaps $G_\mathrm{in} = \{g_i\}_{i=1}^{k}$, where k is the number of gaps. All gaps have the same vertical length and the same horizontal length.

A gap $g \in G_\mathrm{in}$ is a rectangle routing area whose vertical and horizontal lengths are G_v and G_h $(= C_\mathrm{h})$, respectively, and the lower left corner is $(0, y(g))$. The y-coordinates of gaps are given so that gaps are not overlapped and are inside the channel, and the sum of the vertical length of gaps is smaller than the vertical length of the channel.

As an input of GGCR, a set of nets $N_\mathrm{in} = \{n_i\}_{i=1}^{m}$ are given, where m is the number of nets. A net n consists of $r(n)$ pins, i.e., $n = \{p_i\}_{i=1}^{r(n)}$. The pins are connected by wires of width $w(n)$. The coordinate of a pin p is given by $(x(p), y(p))$. The horizontal range of trunk $T(n)$ of net n is defined as $I_\mathrm{x}(n) = [x_\mathrm{min}(n), x_\mathrm{max}(n)]$, where $x_\mathrm{min}(n)$ and $x_\mathrm{max}(n)$ indicate the leftmost and the rightmost x-coordinates of pins in n.

In GGCR, 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 n is allocated to (g, s), the y-coordinate of the trunk of net n is given by $y(n, g, s) = y(g) + s + w(n)/2$, and then the vertical range of the trunk is defined as $I_\mathrm{y}(n, g, s) = [y(g) + s, y(g) + s + w(n)]$. In a feasible allocation, either the intersections of vertical or horizontal ranges of any distinct trunks have to be empty. The vertical wirelength of net n allocated to (g, s) is defined as $l(n, g, s) = \sum_{p \in n} |y(n, g, s) - y(p)|$. Also, the vertical wirelength of net n allocated at y is given by $l_\mathrm{v}(n, y) = \sum_{p \in n} |y - y(p)|$.

Let $a\colon n \mapsto (g^a(n), s^a(n))$ be the allocation function of trunks of nets to gaps where $g^a\colon N_\mathrm{in} \to G_\mathrm{in}$ and $s^a\colon N_\mathrm{in} \to [0, G_\mathrm{v}]$ are the gap allocation and the offset allocation, respectively. Note that the y-coordinate of the trunk of n allocated by a is $y(a(n)) = y(n, g^a(n), s^a(n))$.

Density of a set of net N at x is the sum of widths of trunks of nets whose horizontal range contain x, and is denoted by $d(x, N) = \sum_{n \in \{n \in N | x \in I_\mathrm{x}(n)\}} w(n)$. The density of N is defined as $D(N) = \max_{x \in [0, C_\mathrm{h}]} d(x, N)$.

The objective of GGCR is to allocate all nets into gaps without overlap so as to minimize the wirelength.

$$
\begin{aligned}
\min \quad & \sum_{n \in N_\mathrm{in}} l(n, g^a(n), s^a(n)), \\
\text{s.t.} \quad & g^a(n) \in G_\mathrm{in},\ s^a(n) + w(n) \le G_\mathrm{v}, \forall n \in N_\mathrm{in}, \\
& I_\mathrm{x}(n) \cap I_\mathrm{x}(n') = \emptyset \text{ or} \\
& I_\mathrm{y}(n, g^a(n), s^a(n)) \cap I_\mathrm{y}(n', g^a(n'), s^a(n')) = \emptyset, \forall n, n' \in N_\mathrm{in}.
\end{aligned}
$$

3. Related Works

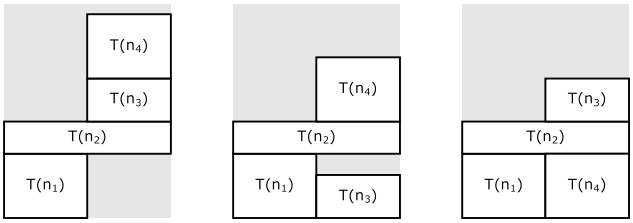
The gridless channel routing [4], [10] has been a subject of researches for a long time in conventional channels. The gridless approach allows an arbitrary location of terminals, nets, and vias, gaining popularity as well as approaches using fine grids in practical situations.

There are variations of gridless channel routing works: two layer [1], [7], multi-layer [2], [8], and routing in real geometries [9]. NLEA by B. Krishna et al. is one of the closest works that deals with various width wires without dogleg and vertical constraint [6]. NLEA allocates trunks to lower left as much as possible with priority given to longer trunks among leftmost trunks that keep the right boundary of allocated region to the left as much as possible.

However, these works are targeted to problems on a single channel, whereas our work targets GGCR, in which the channel is divided into fixed-width gaps. GGCR is difficult because the allocation target is multiple gaps instead of a single channel.

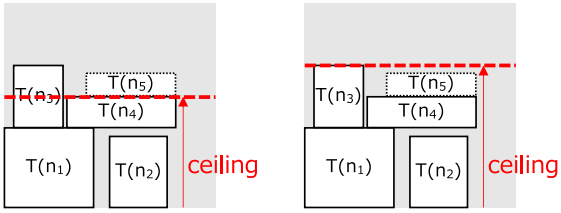
There are few works for GGCR. BCA [16] and enhanced algorithms of BCA prioritize nets to minimize the wirelength based on the relative positions of tracks and pins, but it is intended for single width nets only and cannot be used for wide-width nets. CAP [12] allocates the trunk of a net repeatedly so that each gap is filled as much as possible by adopting an appropriate order of allocation, but CAP is intended only to minimize the number of gaps, not to reduce the wirelength. As a post-processing method, Gap Swap-Flip (GSF) that modifies a given initial trunk allocation to reduce the total vertical wirelength is proposed [11], but it is expected that the wirelength can be reduced more if a better initial solution is given.

This paper proposes CCAP, which minimizes the wirelength for GGCR. CCAP achieves a better chip design by shortening



(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).

Fig. 2 Allocations by different orders.



(a) $T(n_5)$ violates CC. (b) $T(n_5)$ meets CC.

Fig. 3 Ceiling constraint (CC).

the wirelength with the use of given gaps in GGCR.

4. Baseline Algorithm

This section explains ceiling-and-packing (CAP) [12] on which proposed CCAP is based. In order to explain the proposed CCAP, an explanation of CAP is necessary because CCAP differs only in the gap order and the priority of nets from CAP.

4.1 Allocation Scheme

GGCR is a problem of allocating trunks on a two-dimensional plane without overlap in which the x-coordinate of each trunk is fixed. Proposed algorithm allocates each trunk repeatedly as low as possible while avoiding jumping over other trunks allocated so far when it is dropped from above.

The set of nets in N whose trunks contain x is denoted by $M(x, N) = \{n \in N \mid x \in I_x(n)\}$. The set of nets in N whose trunks horizontally overlap with trunk $T(n)$ of net n is denoted by $M(n, N) = \{n' \in N \mid I_x(n') \cap I_x(n) \neq \emptyset\}$. Then, the y-coordinate of the bottom of the trunk of the net is set to $s^a(n) = \max_{n' \in M(n,N)}(s^a(n') + w(n'))$ 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.

Figure 2 gives examples of allocation results of the set of four trunks $T(N_{\text{in}}) = \{T(n_i)\}_{i=1}^4$ in different allocation orders. In Fig. 2 (a), $T(n_3)$ is allocated on the top of $T(n_2)$ even if there is enough space below $T(n_2)$. The allocation order could be more optimal regarding the width used. An optimal result is shown in Fig. 2 (c). Note that there are various optimal results regarding width used and that allocation orders that derive an optimal result are not unique.

4.2 Ceiling-and-packing Algorithm (CAP)

Ceiling-and-Packing algorithm (CAP) [12] is used as the baseline of our proposal. CAP gives a higher priority to wider trunks in allocation as first-fit-decreasing adopts in Bin-Packing. CAP allocates trunks repeatedly from left to right 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 1. First, the function "CAPSort(N_{in})" sorts nets according to the width of the trunks of nets, and then ties are broken by the leftmost principal.

The iteration of allocation of trunks along a horizontal direction from left to right is referred to as *round*. In each round, trunks are allocated from left to right with no overlap. 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. 2 (a)). CAP makes sure that all trunks allocated in the same round do not horizontally overlap each other. The trunk of a net violates the horizontal constraint (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 10 in CAP 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. CAP skips allocation of a higher priority net if HC is violated. 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.

CAP also skips allocation if either zone constraints (ZC) or ceiling constraint (CC) is violated. The details of ZC and CC are explained in the following subsections.

4.3 Zone Constraint (ZC)

In each round, CAP allocates trunks so that the density of nets not allocated so far is reduced as much as possible. The critical zone Z for a set of nets N is defined as $Z = \{x \mid d(x, N) = D(N)\}$. CAP allocates a trunk only when all the critical zone to the left of the trunk is covered by trunks allocated in the round so far. CAP skips allocation if there is the critical 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.

4.4 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 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.

Figure 3 illustrates the effects of CC. $T(n_5)$ violates CC if the ceiling is set to the top boundary of $T(n_4)$ as shown in Fig. 3 (a). If the ceiling is set to the top boundary of $T(n_3)$, then $T(n_5)$ meets CC and is allocated on the top of $T(n_4)$ as show in Fig. 3 (b).

5. Proposal: Criticality-based CAP (CCAP)

We propose criticality-based CAP (CCAP) to shorten wirelength in GGCR. CCAP is based on CAP and introduces a new gap order and a new net priority: 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 a net whose wirelength will be smaller if

Algorithm 1 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
  $g \leftarrow \text{delete}(G)$
  $N' \leftarrow \emptyset$, $C \leftarrow (G_{\text{v}})$
  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

Algorithm 2 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
  $N_{\text{unit}} \leftarrow \{n \in N \mid w(n) = w_{\text{target}}\}$
  $g \leftarrow \text{CGO}(G, N_{\text{unit}})$
  $G \leftarrow \text{delete}(g, G)$
  $N' \leftarrow \emptyset$, $C \leftarrow (G_{\text{v}})$
  $N \leftarrow \text{CriticalitySort}(N, g, G)$
  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

it is allocated to the currently processing gap than to the remaining gaps. Incorporating them with CAP reduces the wirelength significantly.

The pseudo code of CCAP is given in Algorithm 2. 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 nets to the target gap until no net can be allocated under HC, ZC, and CC. The details of CGO and criticality-based net priority are explained in the following subsections.

5.1 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 GGCR, 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 n is allocated to the center of a gap, i.e.,

$$g_{\text{best}}(n, G) = \arg\min_{g \in G} l_{\text{v}}(n, y_{\text{mid}}(g)), \tag{1}$$

where $y_{\text{mid}}(g)$ is the y-coordinate of the center of gap g, i.e., $y_{\text{mid}}(g) = y(g) + \frac{G_{\text{v}}}{2}$. The number of gaps where net n is al-

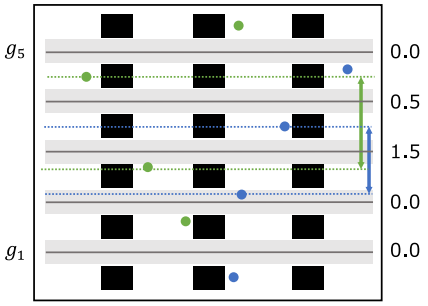


Fig. 4 Gap congestion.

located in the minimum wirelength is represented by $\mathrm{C}(n, G) = |g_{\mathrm{best}}(n, G)|$.

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

$$\mathrm{GC}(g, G, N) = \sum_{n \in \{n' \in N \mid g \in g_{\mathrm{best}}(n', G)\}} \frac{w(n)}{\mathrm{C}(n, G)}. \quad (2)$$

The set of the least congested gaps based on the gap congestion is called *candidate gap* $g_{\mathrm{candidate}}$ and is represented by

$$g_{\mathrm{candidate}} = \arg\min_{g \in G} \mathrm{GC}(g, G, N). \quad (3)$$

If $|g_{\mathrm{candidate}}| > 1$, then algorithm selects an arbitrary gap in candidates as the target gap. In this paper, the lowest positioned gap is selected as target gap, i.e., $g_{\mathrm{target}} = \arg\min_{g \in g_{\mathrm{candidate}}} y_{\mathrm{min}}(g)$.

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 $g_{\mathrm{candidate}} = \arg\min_{g \in G} \mathrm{GC}(g, G, N_{\mathrm{unit}})$, where N_{unit} is the set of the minimum width nets in the set of nets N. In this paper, the minimum width is assumed to be 1, so $N_{\mathrm{unit}} = \{n \in N \mid w(n) = 1\}$.

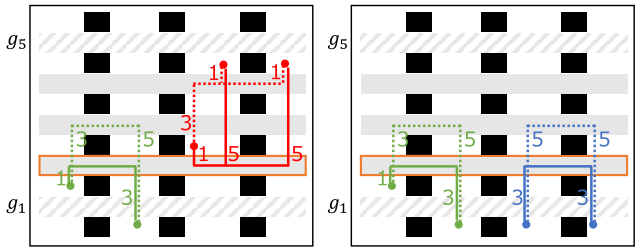
Figure 4 shows an example of gap congestion with 2 nets which have 4 pins and 5 gaps $\{g_i\}_{i=1}^{5}$. The solid gray lines indicate the center of gaps. In the minimum wirelength, green net n_{green} can be allocated to the center of g_3 and g_4, and blue net n_{blue} can be allocated to the center 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 Criticality-based Net Priority

This paper proposes criticality-based net priority to allocate as many nets as possible in as small 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 $x_{\mathrm{min}}(n)$. 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 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



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

Fig. 5 Examples of criticality-based net priority: gray striped boxes indicate gaps where nets have already been allocated; gray boxes indicate remaining gaps and the gray box surrounded by orange lines indicates the target gap.

$$l_{\mathrm{c}}(n, g_{\mathrm{best}}, g_{\mathrm{target}}) = l_{\mathrm{v}}(n, y_{\mathrm{mid}}(g_{\mathrm{best}})) - l_{\mathrm{v}}(n, y_{\mathrm{mid}}(g_{\mathrm{target}})). \quad (4)$$

Note that g_{best} is an arbitrary element of $g_{\mathrm{best}}(n, G)$ (Eq. (1)).

Criticality-based net priority is defined by

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

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

Figure 5 shows two examples of criticality-based net priority where the set of used gaps is $\{g_1, g_5\}$, $g_{\mathrm{target}} = g_2$, and the set of remaining gaps is $\{g_3, g_4\}$. In the case of Fig. 5 (a), if n_{green} is not allocated to the target gap g_2, the wirelength will increase. Thus, n_{green} has a high priority ($l_{\mathrm{c}}(n_{\mathrm{green}}, g_3, g_2) = 4$). 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 ($l_{\mathrm{c}}(n_{\mathrm{red}}, g_4, g_2) = -6$). When two nets have the same priority as shown in Fig. 5 (b), they are prioritized according to the third criterion, the leftmost first order.

5.3 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 $O(km)$ 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 $O(m^3)$.

6. Experiments

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

The first comparison experiment used three algorithms to compare the wirelength: Left-Edge (LE) [3], CAP [12], 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 wirelength: random, bottom-up, top-down, and four variations of CGO. CGO is divided into four kinds: use of all width nets or of only minimum width nets for gap congestion, 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, $G_v = 10$, and horizontal length, $G_h = 1$, 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 $x(p) \sim \mathcal{U}(0,1)$, $y(p) \sim \mathcal{U}(0, C_v), \forall p \in n$, where $C_v = |G_{in}| \times G_v + (|G_{in}| + 1) \times 10$. The width of each trunk w was selected among 1, 2, 3, and 4 according to the probabilities given in **Table 1**.

Table 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

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

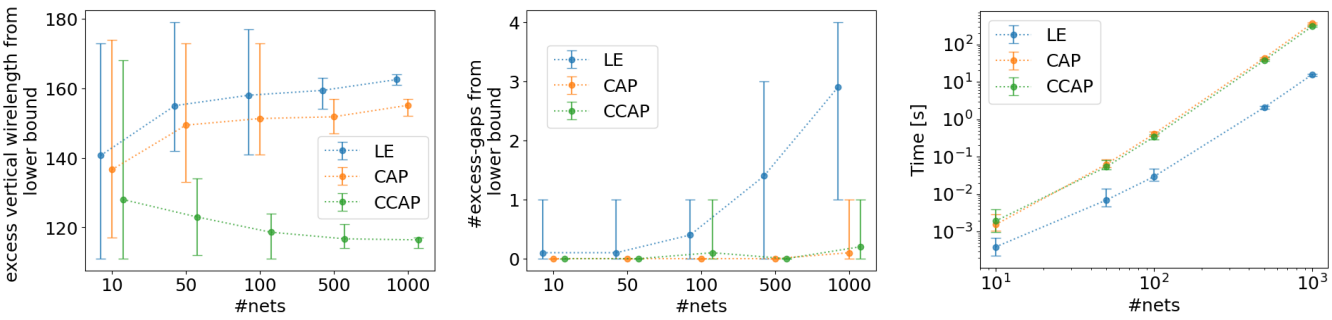
Table 2 shows comparison results in vertical wirelength. Note that there were gaps not used by CAP and CCAP since the chip size was adjusted to the number of gaps LE uses. In the c1 benchmarks, CCAP reduced the wirelength by 10–45 points from the wirelength by CAP. In the c2 benchmarks with net counts 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 2 use only one benchmark for each condition. So, the variations of results were investigated. **Figure 6** shows the variation of allocation results with 10 c1 benchmarks including the benchmarks in Table 2 that are generated with different random seeds. The points shown in each figure are the average value of corresponding metrics, and the errorbars represent the minimum and maximum values of the metrics.

Figure 6 (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 three algorithms 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.

Table 2 Vertical wirelength for $G_v = 10$ and gap separation=10.

Netlist name	#net	Total #pin $\lvert P_{in}\rvert$	Density $D(N_{in})$	#gap	Lower Bound wirelength $x/C_h/\lvert N_{in}\rvert$	$y/C_v/\lvert P_{in}\rvert$ (%)	LE #gap	LE wirelength	CAP #gap	CAP wirelength	CCAP (ours) #gap	CCAP (ours) wirelength
c1-b1	10	51	14	2	0.67	0.2436 (100)	2	0.3472 (143)	2	0.3275 (134)	2	0.2873 (118)
c1-b2	50	258	62	7	0.66	0.2103 (100)	7	0.3214 (153)	7	0.2960 (141)	7	0.2763 (131)
c1-b3	100	519	118	12	0.61	0.2058 (100)	12	0.3254 (158)	12	0.3231 (157)	12	0.2313 (112)
c1-b4	500	2,530	580	58	0.63	0.2101 (100)	59	0.3227 (154)	58	0.3083 (147)	58	0.2415 (115)
c1-b5	1,000	4,924	1,151	116	0.63	0.2048 (100)	117	0.3305 (161)	116	0.3204 (156)	116	0.2367 (116)
c2-b1	10	51	19	2	0.67	0.2436 (100)	2	0.3416 (140)	2	0.3034 (125)	2	0.3034 (125)
c2-b2	50	258	81	9	0.66	0.2103 (100)	9	0.3235 (154)	9	0.3194 (152)	9	0.2747 (131)
c2-b3	100	519	152	16	0.61	0.2058 (100)	16	0.3277 (159)	16	0.3319 (161)	16	0.2575 (125)
c2-b4	500	2,530	777	78	0.63	0.2101 (100)	79	0.3227 (154)	78	0.3313 (158)	78	0.2491 (119)
c2-b5	1,000	4,924	1,555	156	0.63	0.2048 (100)	159	0.3315 (162)	156	0.3415 (167)	157	0.2389 (117)



(a) Excess vertical wirelength from lower bound. (b) Number of excess gaps from lower bound. (c) Computation time.

Fig. 6 Variation in results with 10 different c1 benchmarks.

Table 3 Vertical wirelength with various kinds of gap orders in CCAP.

Netlist						GCF	GCA	GCF	GCA
name	#net	#gap	Random	Bottom-up	Top-down	with N_{in}	with N_{in}	with N_{unit}	with N_{unit}
c1-b1	10	2	0.2873 (118)	0.2873 (118)	0.2811 (115)	0.2811 (115)	0.2873 (118)	0.2811 (115)	0.2873 (118)
c1-b2	50	7	0.2412 (115)	0.2420 (115)	0.2521 (120)	0.2503 (119)	0.2763 (131)	0.2503 (119)	0.2763 (131)
c1-b3	100	12	0.2590 (126)	0.2721 (132)	0.2608 (127)	0.2552 (124)	0.2330 (113)	0.2571 (125)	0.2313 (112)
c1-b4	500	58	0.2468 (117)	0.2638 (126)	0.2702 (129)	0.2600 (124)	0.2387 (114)	0.2597 (124)	0.2387 (114)
c1-b5	1,000	116	0.2363 (115)	0.2665 (130)	0.2608 (127)	0.2556 (125)	0.2334 (114)	0.2561 (125)	0.2337 (114)
c2-b1	10	2	0.3034 (125)	0.3034 (125)	0.3276 (134)	0.3276 (134)	0.3034 (125)	0.3276 (134)	0.3034 (125)
c2-b2	50	9	0.2961 (141)	0.2677 (127)	0.2751 (131)	0.2541 (121)	0.2732 (130)	0.2919 (139)	0.2747 (131)
c2-b3	100	16	0.2690 (131)	0.2939 (143)	0.2761 (134)	0.2924 (142)	0.2619 (127)	0.2917 (142)	0.2575 (125)
c2-b4	500	78	0.2467 (117)	0.2989 (142)	0.3000 (143)	0.2789 (133)	0.2423 (115)	0.2759 (131)	0.2493 (119)
c2-b5	1,000	156	0.2456 (120) [a]	0.3027 (148)	N/A	0.2776 (136)	N/A	0.2747 (134)	N/A

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

Figure 6 (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.

Figure 6 (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 $m = 100$ and 1,000, are $O\left(m^{2.31}\right)$, $O\left(m^{2.59}\right)$, and $O\left(m^{2.67}\right)$, respectively, where m is the number of nets.

Table 3 shows the effect of gap order on the wirelength by using the same benchmarks used in Table 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 order of the gaps.

7. Conclusion

CCAP that minimizes the wirelength in GGCR was proposed. 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 by using almost the same number of gaps as LE and CAP. Since an efficient solution of GGCR is necessary to realize a high-performance chip design, CCAP contributes to these chip designs.

References

[1] Chen, H. and Kuh, E.: Glitter: A Gridless Variable-Width Channel Router, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.5, No.4, pp.459–465 (online), DOI: 10.1109/TCAD.1986.1270217 (1986).

[2] Groeneveld, P.: 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 (online), DOI: 10.1109/43.62773 (1990).

[3] Hashimoto, A. and Stevens, J.: Wire Routing by Optimizing Channel Assignment within Large Apertures, *Proc. 8th Design Automation Workshop* (*DAC*), pp.155–169 (online), DOI: 10.1145/800158.85069 (1971).

[4] Hightower, D. and Boyd, R.: A Generalized Channel Router, *Proc. 17th Design Automation Conference* (*DAC*), pp.12–21 (online), DOI: 10.1145/800139.804507 (1980).

[5] Kobayashi, S., Tashiro, K., Minemura, Y., Nakagami, K., Arita, K., Oohashi, T., Funayama, K., Sakai, H., Mushiga, M., Okabe, K., Kanno, Y., Shimizu, S., Fujikura, E., Nakae, A., Yamaguchi, K., Yamawaki, H., Nakajima, K. and Sato, M.: High Peformance 3D Flash Memory with 3.2Gbps Interface and 205MB/s Program Throughput based on CBA (CMOS Directly Bonded to Array) Technology, *Proc. International Electron Devices Meeting* (*IEDM*) (online), DOI: 10.1109/IEDM45741.2023.10413716 (2023).

[6] Krishna, B., Chen, C. and Sehgal, N.: Routing wires with non-uniform width and spacing in data paths, *Proc. 11th International Conference on Microelectronics* (*ICM*), pp.85–88 (online), DOI: 10.1109/ICM.2000.884811 (1999).

[7] Ng, C.H.: A "gridless" variable-width channel router for marco cell design, *Proc. 24th Design Automation Conference* (*DAC*), pp.633–636 (online), DOI: 10.1145/37888.37988 (1987).

[8] Polkl, D.B.: A three-layer gridless channel router with compaction, *Proc. 24th Design Automation Conference* (*DAC*), pp.146–151 (online), DOI: 10.1145/37888.37910 (1987).

[9] Rothermel, H.-J. and Mlynski, D.: Automatic Variable-Width Routing for VLSI, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.2, No.4, pp.271–284 (online), DOI: 10.1109/TCAD.1983.1270045 (1983).

[10] Sato, K., Shimoyama, H., Nagai, T., Ozaki, M. and Yahara, T.: A "Grid-free" Channel Router, *Proc. 17th Design Automation Conference* (*DAC*), pp.22–31 (online), DOI: 10.1145/800139.804508 (1980).

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

[12] Shimoda, M. and Takahashi, A.: Gridless Gap Channel Routing with Variable-width Wires, *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, Vol.108, No.3 (online), DOI: 10.1587/transfun.2024VLP0003 (2025). (in press).

[13] Tagami, M.: CMOS Directly Bonded to Array (CBA) Technology for Future 3D Flash Memory, *Proc. International Electron Devices Meeting* (*IEDM*) (online), DOI: 10.1109/IEDM45741.2023.10413718 (2023).

[14] Taniguchi, K., Tayu, S., Takahashi, A., Molongo, M., Minami, M. and Nishioka, K.: Two-layer Bottleneck Channel Track Assignment

for Analog VLSI, *IPSJ Trans. System and LSI Design Methodology*, Vol.17, pp.67–76 (online), DOI: 10.2197/ipsjtsldm.17.67 (2024).
[15] Taniguchi, K., Tayu, S., Takahashi, A., Molongo, M., Minami, M. and Nishioka, K.: A Fast Three-layer One-side Bottleneck Channel Routing with Layout Constraints using ILP, *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, Vol.108, No.3 (online), DOI: 10.1587/transfun.2024VLP0002 (2025). (in press).
[16] Wang, Z., Shimoda, M. and Takahashi, A.: BCA Channel Routing to Minimize Wirelength for Generalized Channel Problem, *Proc. IEEE International Symposium on Circuits and Systems* (*ISCAS*) (online), DOI: 10.1109/ISCAS58744.2024.10558428 (2024).

Masayuki Shimoda received his B.E. and M.E. degrees in engineering from Tokyo Institute of Technology, Tokyo, Japan, in 2018 and 2020, respectively. He is currently a Doctoral student with the Department of Information and Communications Engineering of Tokyo Institute of Technology. His current research interests include machine learning, VLSI physical design, and computer architecture.

Atsushi Takahashi received his B.E., M.E., and D.E. degrees in electrical and electronic engineering from Tokyo Institute of Technology, Tokyo, Japan, in 1989, 1991, and 1996, respectively. He was at the Tokyo Institute of Technology as a Research Associate from 1991 to 1997, and as an Associate Professor from 1997 to 2009 and from 2012 to 2015. From 2009 to 2012, he was at Osaka University, Suita, Japan, as an Associate Professor. He is currently a Professor with Department of Information and Communications Engineering, School of Engineering, Tokyo Institute of Technology. His current research interests include VLSI layout design and combinational algorithms. He is a senior member of IEEE and IPSJ, and a member of ACM.

(Recommended by Associate Editor: *Kohei Miyase*)