A DAG-Based Algorithm for Obstacle-Aware Topology-Matching On-Track Bus Routing *

Chen-Hao Hsu[1], Shao-Chun Hung[2], Hao Chen[2], Fan-Keng Sun[2], and Yao-Wen Chang[1,2]
[1]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 10617, Taiwan
[2]Department of Electrical Engineering, National Taiwan University, Taipei 10617, Taiwan
chhsu@eda.ee.ntu.edu.tw; {b03901019, b03901016, b03901056, ywchang}@ntu.edu.tw

ABSTRACT

As clock frequencies increase, topology-matching bus routing is desired to provide an initial routing result which facilitates the following buffer insertion to meet the timing constraints. Our algorithm consists of three main techniques: (1) a bus clustering method to reduce the routing complexity, (2) a DAG-based algorithm to connect a bus in the specific topology, and (3) a rip-up and re-route scheme to alleviate the routing congestion. Experimental results show that our proposed algorithm outperforms all the participating teams of the 2018 CAD Contest at ICCAD, where the top-3 routers result in 145%, 158%, and 420% higher costs than ours.

CCS CONCEPTS

• **Hardware → Electronic design automation**;

KEYWORDS

Physical Design, Bus Routing, Topology Matching, Advance Technologies

1 INTRODUCTION

As the semiconductor industry evolves into the nanometer era, the scale of modern electronic systems grows rapidly. The number of signal nets in a circuit also increases significantly, making manual routing an extremely complicated task. In modern VLSI designs, *buses* are widely used to transfer parallel information between functional units. The increasing clock frequency also induces special design constraints for bus routing (e.g., the *length-matching constraint* [9, 12, 14] and the *exact-matching constraint* [10, 11]). Traditional bus routers often cannot handle these constraints well. Therefore, it is desirable to develop an effective algorithm for bus routing with modern design constraints.

Timing skew is a common routing constraint in analog/mixed-signal circuit and printed circuit board (PCB) bus designs, where data transferred in all bits of a bus should arrive at the destination pins almost simultaneously. To meet the timing-skew constraint, the length of all the wires of the same bus should be approximately the same. As a result, existing frameworks considering the length-matching constraint (i.e., the length differences of all bits of the same bus are within specific bounds) have been proposed. Ozdal *et al.* [13] proposed a length-matching algorithm for high-performance PCBs, ensuring that both maximum and minimum length constraints are satisfied. Yan *et al.* [15] presented a length-matching gridless router for general routing topology, so as to deal with practical designs without topology restriction efficiently.

As advanced node enablement such as N7 and N5 being deployed in modern semiconductor process technology, the traditional uniform routing track configuration is not sufficient for modern circuit designs. As a result, the non-uniform track configuration with an irregular track arrangement and different width constraints is utilized. In addition, more complicated design rules (e.g., min-area rules, corner-to-corner spacing rules, etc.) become necessary. Due to the new challenges of bus routing, previous length-matching frameworks might not be sufficient for meeting real-world IC design requirements and generating routing results without design rule violations. So bus routing algorithms considering the topology-matching constraint are preferred as they provide higher flexibility in timing optimization. If all the bits of a bus are routed in an isomorphic topology and thus *topology-matching routing*, it is often easier for the following timing-aware refinement stages (e.g., buffer insertion) to meet the timing requirements. Figure 1 shows the bus routing results considering the length-matching and topology-matching constraints.

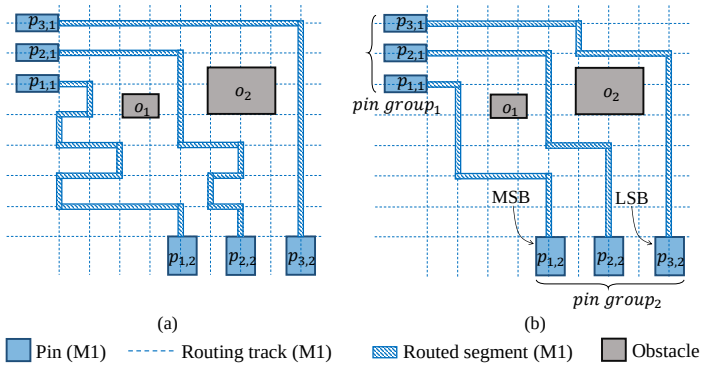

Figure 1: (a) A bus routing that meets the length-matching constraint. (b) A bus routing that satisfies the topology-matching constraint. $p_{i,j}$ denotes the j^{th} pin of the i^{th} bit in a bus, and o_i denotes the i^{th} obstacle in the design.

However, not much work addressed the topology-matching bus routing problem [8]. To the best of our knowledge, further, no published work presents a completed topology-matching bus routing flow with real-world requirements such as the non-uniform track configuration. To stimulate the research on this topic, the 2018 CAD Contest at ICCAD [2] held the first contest on a topology-matching bus routing problem.

Based on the Contest, we present a robust topology-matching bus router for advanced VLSI technologies. Our proposed algorithm solves the designated topology-matching bus routing problem while considering some modern design constraints. The main contributions of this paper are summarized as follows:

- We propose an efficient bus clustering algorithm to group the buses with the same pin topology, which can substantially reduce the bus routing complexity.
- We develop a novel DAG-based topology-matching on-track bus routing algorithm that connects all the bits of a bus in a specific topology while handling the spacing rules and non-uniform track configuration.
- We design a rip-up and re-routing scheme that alleviates routing congestion, which significantly reduces the number of spacing violations.
- Experimental results show that our router achieves much better routing quality than any other participating router at the 2018 CAD Contest at ICCAD. For example, the top-3 routers of the contest result in 145%, 158%, and 420% higher costs than ours.

The remainder of this paper is organized as follows. Section 2 introduces basic elements of buses and the design constraints used in the contest [2] and formulates the obstacle-aware on-track topology-matching bus routing problem. Section 3 details the core techniques of our algorithm. Section 4 shows the experimental results, and Section 5 concludes this paper.

*This work was supported by AnaGlobe, TSMC, MOST of Taiwan under Grant No's MOST 105-2221-E-002-190-MY3, MOST 106-2911-I-002-511, and MOST 107-2221-E-002-161-MY3.

DAC '19, June 2–6, 2019, Las Vegas, NV, USA

ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317740

2 PRELIMINARIES

In this section, we introduce some bus basics, the non-uniform track configuration, the topology-matching constraints and the definition of compactness of a bus. Then, we formulate the obstacle-aware topology-matching on-track bus routing problem.

2.1 Bus Basics

Here, we give the terminology of a bus used in this paper, including *pin group*, *bit*, *segment*, and *wire width*.

2.1.1 Pin Group. A *pin group* is a group of aligned pins in a bus, where the x- or y-coordinates of the center points of pins are identical. A pin group is vertical (horizontal) if the aligned pins have the same x-coordinates (y-coordinates). Besides, the *pin group position* is defined as the x-coordinate (y-coordinate) of its pin for a vertical (horizontal) pin group. Moreover, the order of the pins within a pin group is the same or reverse bit order from LSB to MSB. Finally, the number of pins in each pin group in a bus is exactly the same.

2.1.2 Bit. For each pin in a pin group, there exists a corresponding pin in every other pin group of the bus, which should be connected together. The set of these relevant pins is defined as a *bit*. Hence, the number of bits exactly matches the number of pins in a pin group. Once all the bits in a bus are connected, the bus is considered as connected.

2.1.3 Segment. A *segment* is a set of matched wires of each bit in a topology-matching routing result. Thus, the number of wires in a segment is equal to the number of bits.

2.1.4 Wire Width. For a bus, the *wire widths* of different layers may be different. Typically, the wire width in a higher layer is larger.

Figure 1(b) illustrates a bus with two pin groups, three bits, and four segments. Note that the bus in Figure 1(b) satisfies the topology-matching constraint, to be detailed in Section 2.3.

2.2 Non-Uniform Track Configuration

Routing tracks are important for helping a router to handle design rules and mask coloring for multiple patterning [4]. To meet the complex routing requirements for different buses, a simple uniform track configuration might not be sufficient. So the non-uniform track configuration is adopted, which is composed of tracks with different width constraints. It is prohibited to route a wire on a track if the wire width is greater than the width constraint of the track. The spacing between two tracks are non-uniform as well. Figure 2 shows an example of the non-uniform track configuration. Besides, there are even overlapping tracks with different wire constraints in the design. In this case, only the wires with widths less than or equal to the maximum width constraints in the overlap region are allowed to be routed. For example, $t_{3,1}$ and $t_{3,2}$ are overlapped as shown in Figure 2. Any wire routed through the overlap region should have a width satisfying the width constraint of $t_{3,1}$ because it has a larger width constraint.

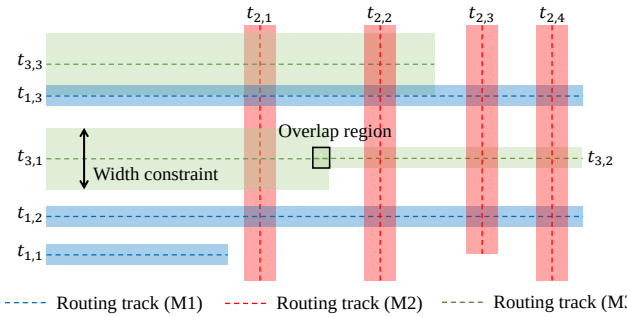

Figure 2: An example of the non-uniform track configuration. $t_{i,j}$ denotes the j^{th} track in the metal layer i.

2.3 Topology-Matching Constraints

To meet the topology-matching constraint, the following five constraints should be satisfied:

- *Wire Count Constraint*: the number of wires of a bit should be the same.
- *Ordering Constraint*: within each segment, the wires of different bits should follow the same or reverse bit order (i.e., LSB to MSB or MSB to LSB).
- *Layer Constraint*: within each segment, the wires of a bit should be in the same layer.
- *Direction Constraint*: within each segment, the wires of a bit should route towards the same direction.
- *T-junction Constraint*: T-junctions should intersect at the same segment.

Figure 3 shows an example of the routing constraints. The bus consists of three 2-pin bits b_1, b_2, and b_3, where the pin set of b_1, b_2, and b_3 is $\{p_{1,1}, p_{1,2}\}$, $\{p_{2,1}, p_{2,2}\}$ and $\{p_{3,1}, p_{3,2}\}$, respectively. In Figure 3(a), all bits are considered to have the same topology. In Figure 3(b), the bit b_3 has six wires, while other bits have only four wires, which violates the wire count constraint. In Figure 3(c), if we trace from the upper-left pin group, the wires within the second segment are in the wrong ordering (i.e., neither in the same nor reverse bit order), which violates the ordering constraint. The bus in Figure 3(d) violates the direction constraint because the second wire of b_3 routes upwards while the second segment of other bits routes downwards. Finally, a T-junction might occur when a bit has more than two pins. As shown in Figure 3(e), within a segment, if a bit is a T-junction, the other bits should also be T-junctions to satisfy the T-junction constraint.

2.4 Bus Compactness

To maintain the similarity among the bits within a bus, the bus should keep as compact as possible. The compactness of a bus is defined as the summation of the distance between the LSB and MSB of all segments. In Figure 4, the bus in Figure 4(a) is more compact than that in Figure 4(b) because $\sum_{i=1}^{4} w_i < \sum_{i=1}^{4} w'_i$. Despite that the total wirelengths of the bus in Figure 4(a) and Figure 4(b) are the same, the bus in Figure 4(a) is more preferable because it is more compact.

2.5 Problem Formulation

We formally define the obstacle-aware topology-matching on-track bus routing problem below:

Problem 1 (Obstacle-Aware Topology-Matching On-Track Bus Routing Problem). *Given a set of n buses $S_B = \{B_1, \ldots, B_i, \ldots, B_n\}$, and a set of m obstacles $S_o = \{o_1, \ldots, o_j, \ldots, o_m\}$, generate a routing solution for each bus $B_i \in S_B$, such that B_i is on-track and meets the topology-matching constraint, and the following metrics are optimized/minimized: (1) the total wirelength of all buses, (2) the segment count of each bus, (3) the compactness of each bus, and (4) the number of total spacing violations.*

3 PROPOSED ALGORITHMS

The obstacle-aware topology-matching on-track bus routing problem is complex due to the topology-matching constraint for each bus, the congestion problem among the buses, and also the design rules. Therefore, we propose a novel algorithm to handle the bus topology and alleviate the routing congestion as well. Our proposed algorithm consists of two major steps: (1) a bus clustering technique based on the well-known longest common subsequence (LCS) algorithm [5], which merges buses with two pin groups together to reduce the routing complexity and (2) a topology-matching bus routing algorithm that iteratively connects all buses while considering the important metrics mentioned in Section 2. Figure 5 shows the overall flow of the proposed algorithm. The following subsections detail the two stages.

3.1 LCS Bus Clustering

Most buses in modern VLSI designs are buses with two pin groups, and furthermore they usually have similar pin topologies. The bus clustering

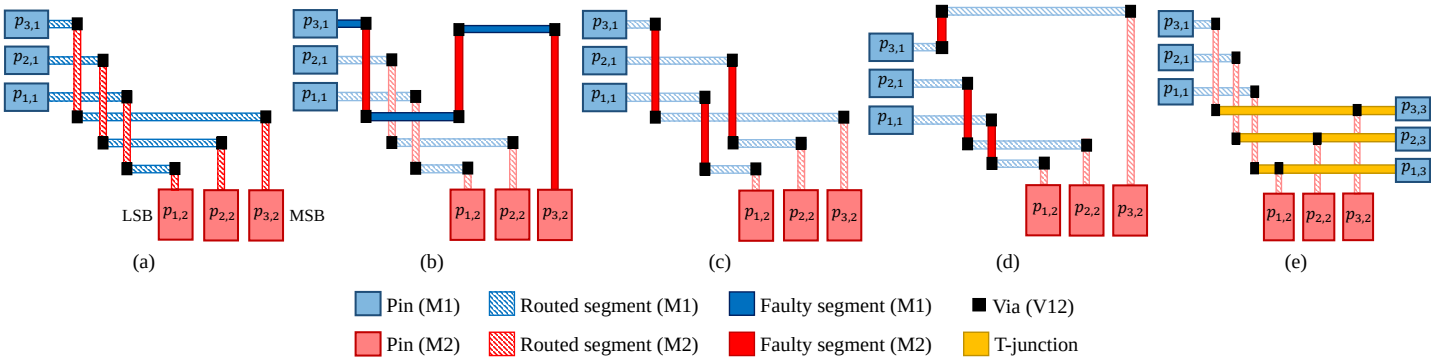

Figure 3: Examples of the five topology constraints. (a) A bus routing meeting all the topology constraints. (b) The third bit violates the wire count constraint. (c) The second segment violates the ordering constraint. (d) The second segment violates the direction constraint. (e) A routing with T-junctions.

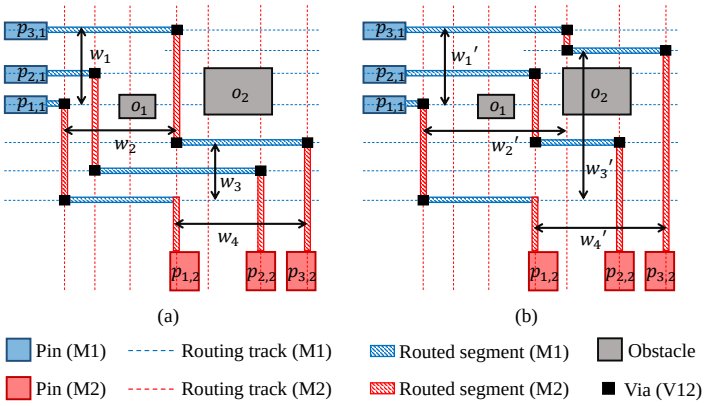

Figure 4: (a) An example of a well-compacted bus. (b) A routing with a higher compactness cost.

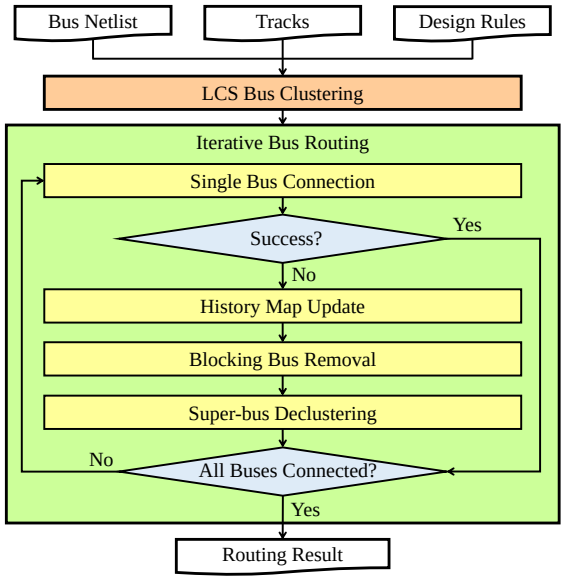

Figure 5: Overview of our algorithm.

technique can be applied to facilitate the routing process. Notably, our bus clustering technique only considers buses with two pin groups.

Definition 1. *A* super-bus *is composed of two or more buses, where the original buses should meet the following constraints:*

(1) *For each layer, the wire widths for the original buses should be the same.*

(2) *The pin group positions in a bus can respectively match those of the other buses so that the number of pin groups remains unchanged after clustering.*

(3) *The pin orders of the two pin groups should be consistent after clustering in order to meet the topology-matching constraint.*

First, we collect the buses into a clustering candidate list (CCL) if their wire widths for each layer and pin group positions are exactly the same (i.e., Constraint (1) and Constraint (2)). Second, to maximize the number of clustered buses in a CCL, the well-known LCS algorithm is performed to find the longest common subsequence in terms of the bits for each CCL. Then, we cluster the buses whose bits are *all* in the LCS. Besides, the LCS algorithm is performed twice on the two pin sequences in different directions, and the longer subsequence will be selected for clustering. The buses in the LCS must satisfy Constraint (3) since the two pin groups of the resulting super-bus follow the same bit ordering (i.e., LCS). The clustered buses will be removed from the CCL, and the LCS clustering will be performed on the updated list. This iterative process terminates if no more buses can be clustered.

Figure 6 illustrates an example of the bus clustering technique. The CCL contains B_a, B_b, and B_c, and the LCS shown in Figure 6(a) is $p^a_{1,*}$, $p^a_{2,*}$, $p^a_{3,*}$, $p^b_{2,*}$, $p^c_{1,*}$, $p^c_{2,*}$. The notation $p^k_{i,*}$ denotes the pins of the i^{th} bit in the bus B_k; for example, in Figure 6(a), $p^a_{1,*}$ contains $p^a_{1,1}$ and $p^a_{1,2}$ in the bus B_a. Since all the bits of B_a and B_c in the LCS, B_a and B_c are thus clustered into the super-bus B_d as shown in Figure 6(b).

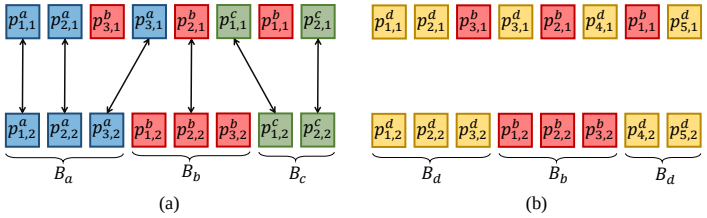

Figure 6: (a) An example shows the buses B_a, B_b, and B_c, and the relative positions of their pins. (b) The super-bus B_d is generated by merging buses B_a and B_c. Here, $p^k_{i,j}$ denotes the j^{th} pin of the i^{th} bit in the bus B_k.

3.2 Iterative Bus Routing

After the bus clustering, the iterative bus routing is performed to connect all the buses. To alleviate the routing congestion in a design, we adopt the negotiation-based rip-up and re-route technique [7] in our bus routing framework. To maintain the routing ordering, all unrouted buses (including super-buses) are pushed into a maximum-priority queue. The cost function used by the priority queue is

$$C_B = \alpha_g \cdot n_g + \alpha_b \cdot n_b + \alpha_f \cdot n_f, \tag{1}$$

where C_B is the total cost of a bus B, n_g is the number of pin groups in B, n_b is the number of bits in B, n_f is the number of continuous failures in the rip-up phase, and α_g, α_b, and α_f are weighted constants. In our implementation, α_g, α_b, and α_f are set to 32, 1, and 8, respectively. Since high-degree buses (i.e., buses with more pin groups) usually need more routing resource and it is much harder to successfully route high-degree buses, we set α_g to a higher value. For buses with many bits, it is also difficult to complete the routing without design rule violations. Therefore, by considering n_b in the cost function, buses with more bits have higher priorities; this way can increase the routing success rate. Moreover, n_f is added to the cost function. If a bus fails for many times, then its priority should increase since the bus might be easily blocked by other previously routed buses.

Once the priority queue is initialized, we can start with the negotiation-based routing scheme, which consists of the four main steps: (1) single bus connection, (2) history map update, (3) blocking bus removal, and (4) super-bus declustering. We detail these four steps below.

Algorithm 1 *RouteSingleBus(D, B)*

Input: D: the input design, B: the target bus.
Output: R_B: the routing result of the bus B.

```
1:  S_b := RandomChooseBits(B, k); // the set of k randomly chosen bits from B
2:  S_g := ∅; // the set of routing guides of the k bits in S_b
3:  for each bit b ∈ S_b
4:      g_b := PathSearch(b, D);
5:      S_g := S_g ∪ {g_b};
6:  PQ_g := TopologyAnalysis(S_g);
7:  while PQ_g is not empty
8:      extract the top guide g_golden from PQ_g;
9:      n_f := 0;
10:     while n_f < n_max
11:         Decision(g_golden);
12:         for each bit b ∈ B
13:             G := ConstructDAG(b, g_golden, D);
14:             if no path from s to t in G
15:                 n_f := n_f + 1;
16:                 R_B := ∅;
17:                 break ;
18:             else
19:                 r_b := ShortestPath(G);
20:                 R_B := R_B ∪ {r_b};
21:                 if B is successfully connected
22:                     return R_B;
23: return ∅;
```

3.2.1 Single Bus Connection. Algorithm 1 shows our topology-matching routing algorithm for a single bus. First, we randomly choose k bits from the target bus B to the set S_b. Then, the A* path searching algorithm is applied to each bit in S_b to generate the routing guides, and the guides are stored in S_g. If k is large, the potential routing results of B presented by the routing guides may be more accurate. However, the runtime of generating routing guides will also increase. To balance the solution quality of routing guides and runtime, we set the value of k by the following equation,

$$k = \min\left(n_{bit}^B, \max\left(10, \min(50, \frac{n_{bit}^B}{4})\right)\right), \tag{2}$$

where n_{bit}^B is the number of bits in bus B. Figure 7 shows an example of generating the routing guides of a three-bit bus.

After generating the routing guides of the selected bits, the topology analysis is performed to collect the guides with the same topology into a *topology candidate list (TCL)*. For each TCL, we choose a guide to be the representative guide for the list. Then all representative guides are pushed into the max-priority queue PQ_g, where the cost used in PQ_g is the number of guides in the corresponding list. In Figure 7, since the topology of g_1 and g_2 are identical, they are pushed into a TCL. No guides share the same topology with g_3, and thus g_3 forms a TCL itself.

The larger size of a TCL means that the topology of its representative guide is more *popular* among the selected bits in S_b. Therefore, we extract

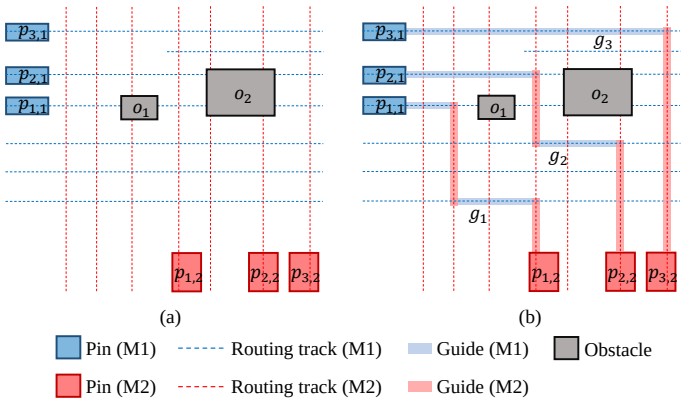

Figure 7: (a) An unrouted bus. (b) The routing guides generated by applying the A* path search algorithm to each bit. g_i represents the guide of the i^{th} bit.

the top guide from PQ_g as the golden guide g_{golden} and try to connect all the bits by following the topology of g_{golden}. For each segment of a guide, there are two *bit directions*. For a horizontal (vertical) segment, the bit direction can be LSB to MSB (denoted as $d_{L\to M}$) or MSB to LSB (denoted as $d_{M\to L}$) in increasing y-coordinate (x-coordinate) order. So we make decisions on the bit direction of each segment in g_{golden} before connecting the bits.

After deciding the bit direction of each guide segment, we route all bits sequentially from LSB to MSB. To connect a bit in the specific topology, a routing directed acyclic graph (DAG) $G(V, E)$ is constructed with several properties. If there exists a path from source s to target t, then the shortest path from s to t is the most compact route for the bit without any spacing violations. Otherwise, if no path exists from s to t, it is impossible to route the bit in the golden topology under the specified searching region. The routing DAG construction consists of three steps: (1) deciding searching region, (2) constructing on-track vertices, and (3) constructing edges with compactness costs.

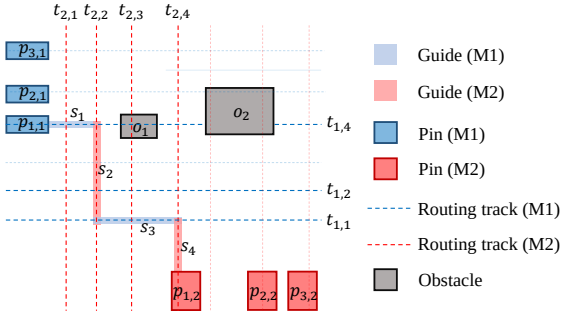

Figure 8: The construction of usable track lists (UTLs).

The size of the searching region impacts the runtime and routing quality. For each segment, we maintain a usable track list (UTL). The formations of UTLs for routing the first bit and non-first bits are different. For the first bit, the n_{first} nearest tracks of the segment s_i for the *both sides* are pushed into the track position list UTL_i. On the other hand, for the other bits, since the previous bit must be connected and the bit direction of each segment needs to be followed, the $n_{non\text{-}first}$ nearest tracks of each wire of the previous bit for *one side* are pushed into the UTL. To achieve a better trade-off between the runtime and routing quality, n_{first} and $n_{non\text{-}first}$ are defined as follows:

$$n_{first} = \min(100, 6n_{bit}^B), \tag{3}$$

$$n_{non\text{-}first} = \min(100, 2n_{bit}^B), \tag{4}$$

where n_{bit}^B is the number of bits in bus B. In Figure 8, suppose we are connecting the first bit and n_{first} is 2, UTL_2 is $\langle t_{2,1}, t_{2,2}, t_{2,3}\rangle$ and UTL_3

is $\langle t_{1,1}, t_{1,2}\rangle$. Moreover, the segment which connects a pin directly or through a via is called a *critical segment*. For a critical segment, only the tracks which overlap with the pin are collected into the UTL. For instance, in Figure 8, s_1 and s_4 are critical segments so UTL_1 and UTL_4 are $\langle t_{1,4}\rangle$ and $\langle t_{2,4}\rangle$, respectively.

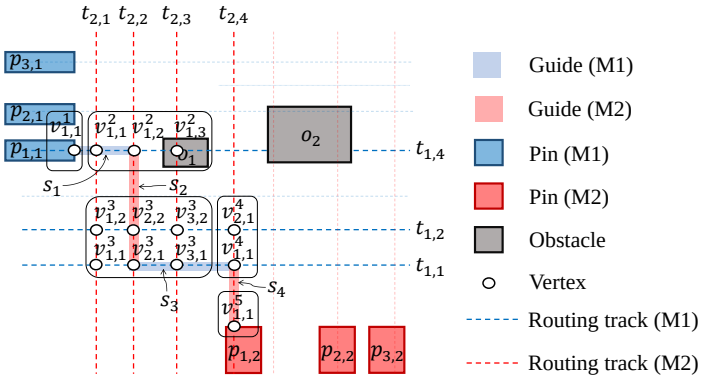

Figure 9: An example of vertex construction. Let $v^k_{i,j}$ denote the vertex in box_k, which is the intersection points of the i^{th} track in UTL_{k-1} and the j^{th} track in UTL_k.

For two adjacent segments, say s_i and s_{i+1}, the intersection points of the perpendicular tracks in UTL_i and UTL_{i+1} form a *box*. For a critical segment, say s_j, the intersection points of the tracks in UTL_j and the pin boundary also form a box. All box points are vertices in G. For example, there are five boxes in Figure 9. The usable tracks of the critical segment s_1 and the pin $p_{3,2}$ form box_1, which consists of $v_{1,1}$. The available tracks of s_1 and s_2 form box_2, which contains three vertices $v^2_{1,1}$, $v^2_{1,2}$, and $v^2_{1,3}$.

If we connect a bit by following the box sequence, the topology of the bit will be the same as the golden one. Therefore, we construct directed edges from the points in box_i to the points in box_{i+1} if the corresponding wires are valid. A wire is valid if it is on-track without spacing violations. In order to make the bus compact, we use the costs of edges to encourage wires to get close to the routed wires of the previous bit. The *track position* of a horizontal (vertical) track is defined as the y-coordinate (x-coordinate) of the track. To determine edge costs, each UTL is sorted by the track positions in increasing (decreasing) order if the bit direction of the corresponding segment is $d_{L\to M}$ ($d_{M\to L}$). Then the usable tracks in each UTL are indexed from 1 according to the sorted order. In Figure 9, suppose that the bit directions of all segments are $d_{L\to M}$. For s_2, the usable tracks $t_{2,1}$, $t_{2,2}$, and $t_{2,2}$ are indexed 1, 2, and 3, respectively. Similarly, for s_3, $t_{1,1}$ and $t_{1,2}$ are indexed 1 and 2, respectively. The cost of an edge is defined by the following equation,

$$\text{cost}\big(e(v^{k-1}_{i_1,j_1}, v^{k}_{i_2,j_2})\big) = i_2 \cdot j_2, \tag{5}$$

where $e(v_1, v_2)$ is a directed edge from v_1 to v_2. $v^k_{i,j}$ is a vertex in box_k and represents the intersection point of the i^{th} track in UTL_{k-1} and the j^{th} track in UTL_k. In Figure 9, the cost of $e(v^2_{1,2}, v^3_{2,2})$ is 4. The edge $e(v^2_{1,3}, v^3_{3,2})$ is not constructed in G due to a spacing violation with the obstacle o_1. Finally, the source node s and the target node t are added to G. The edges from s to all the vertices in the first box are constructed with zero cost, and also the edges from all the vertices in the last box are built with zero cost. Figure 10 shows the routing DAG of the instance in Figure 9.

After the construction of the routing DAG, we apply the well-known Dijkstra shortest path algorithm [6] to find the path with the least cost. If a path is found, then the bit is connected in the golden topology. In Figure 10, the shortest path is $\langle s, v^1_{1,1}, v^2_{1,1}, v^3_{1,1}, v^4_{1,1}, v^5_{1,1}, t\rangle$, and the routing result of the first bit is shown in Figure 11(a). Figure 11 illustrates the routing results if we can connect all the bits successfully. However, if we fail to connect a bit, then all the wires of previous bits are ripped-up. The bit directions are re-decided before re-routing all the bits. When the number of failures reaches the user-defined number n_{max}, it is considered that the bus cannot be connected in the golden topology. Then the top guide of

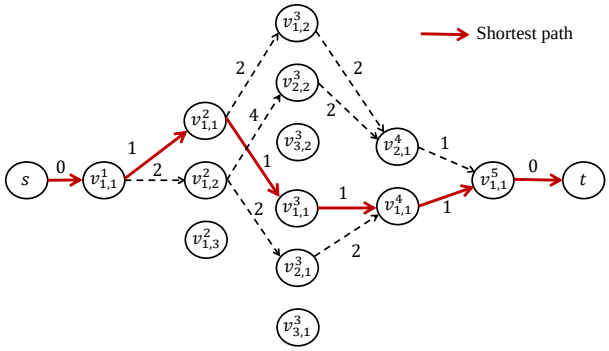

Figure 10: The constructed routing DAG of Figure 9 and the cost of each edge is labelled aside. The solid arrows represent the shortest path from s to t.

PQ_g is extracted to be the next golden guide. As the PQ_g is empty, the bus fails to be connected by following any candidate topology.

3.2.2 History Map Update. Bus connection failures may result from the blocking buses which block the current bus. In the construction of the routing DAG, the blocking buses can be obtained while building the directed edges. To avoid the same routing result, we increase the costs of the blocking buses in the history map after the single bus connection fails. Therefore, in the next iteration, when building the guides of a bus, the router will detour to avoid the congested region due to its higher cost.

3.2.3 Blocking Bus Removal. The blocking buses are ripped-up after the history map is updated. Since the number of blocking buses may be large for some congested cases, the maximum number of ripped-up buses is limited to a user-defined number. In our implementation, at most three buses will be removed in an iteration.

3.2.4 Super-Bus Declustering. The track configuration may result in the routing failure of a super-bus. Thus, after the connection of a super-bus fails a user-defined number of times, the super-bus is split into two buses as evenly as possible.

4 EXPERIMENTAL RESULTS

We implemented our algorithm in the C++ programming language with the Boost C++ libraries 1.67.0 [3] and the Lemon graph library [1]. All experiments were performed on a Linux workstation with 4 Xeon 3.5 GHz CPUs with 72GB memory. The experiments were conducted on the benchmarks from the 2018 CAD Contest at ICCAD, where the benchmark statistics is summarized in Table 1.

Table 1: Benchmark characteristics. #buses, #bits, #pins, #obs, and #layers denote the numbers of buses, total bits, obstacles, and layers, respectively.

Circuit	#buses	#bits	#pins	#obs	#layers
beta_1	34	1260	2520	159	3
beta_2	26	1262	2524	0	3
beta_3	60	665	1330	555108	3
beta_4	62	698	1396	0	3
beta_5	6	1964	3928	0	4
final_1	18	1032	2064	0	3
final_2	70	1285	2570	0	3
final_3	47	852	1704	0	4

To evaluate the solution quality, the contest [2] provides a cost function considering the metrics described in Section 2. The total cost (C_{total}) is the sum of the routing cost (C_r), the spacing violation penalty cost (C_s), and the routing failure penalty cost (C_f), where lower values are better. The routing cost (C_r) is a weighted sum of the following three costs: the normalized wirelength cost (C_{wi}), the normalized segment count cost (C_{si}), and the normalized bus width cost (C_{ci}). The spacing violation

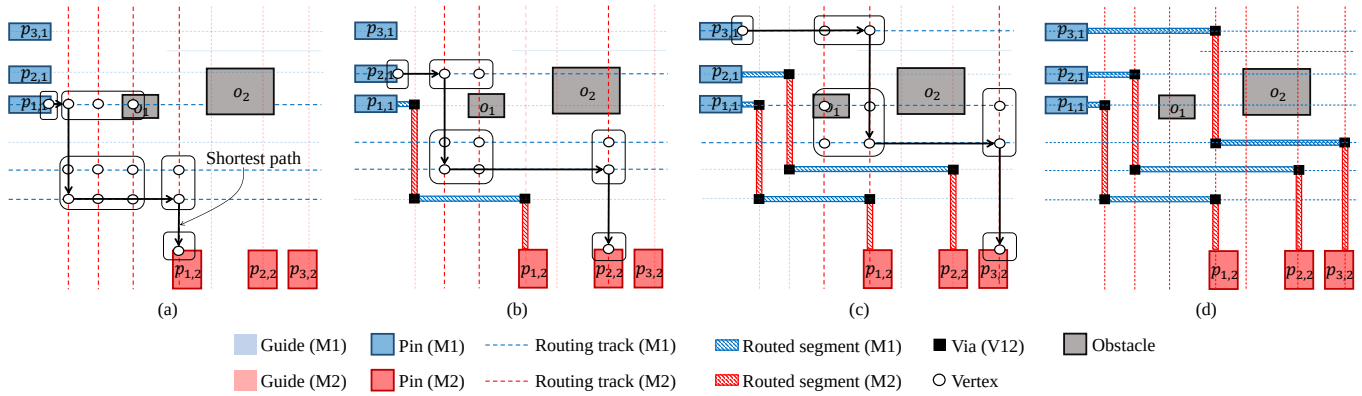

Figure 11: The demonstration of the bus connection bit-by-bit.

Table 2: Comparison of the total routing cost (C_r), the total spacing cost (C_s), the number of spacing violations (n_s), the routing failure cost (C_f), the number of buses with routing failures (n_f), and the total cost (C_{total}). The runtime of our algorithm is given in the last column.

	1st Place						2nd Place						3rd Place						Ours						
Circuit	C_r	C_s	n_s	C_f	n_f	C_{total}	C_r	C_s	n_s	C_f	n_f	C_{total}	C_r	C_s	n_s	C_f	n_f	C_{total}	C_r	C_s	n_s	C_f	n_f	C_{total}	Time (s)
beta_1	688.70	280	35	0	0	968.70	700.52	5096	637	0	0	5796.52	641.48	8744	1093	4000	2	13385.48	685.76	0	0	0	0	685.76	3
beta_2	514.65	760	95	0	0	1274.65	563.15	4904	613	0	0	5467.15	484.15	9472	1184	2000	1	11956.15	521.82	0	0	0	0	521.82	156
beta_3	1936.28	0	0	0	0	1936.28	2024.11	0	0	0	0	2024.11	1998.70	1928	241	0	0	3926.70	1929.77	0	0	0	0	1929.77	71
beta_4	2192.41	0	0	0	0	2192.41	2270.69	0	0	0	0	2270.69	2250.14	1048	131	0	0	3298.14	2176.90	0	0	0	0	2176.90	3
beta_5	118.99	1848	231	0	0	1966.99	94.72	616	77	2000	1	2710.72	98.16	1216	152	2000	1	3314.16	120.72	600	75	0	0	720.72	110
final_1	326.57	830	83	2000	1	3156.57	366.67	2750	275	2000	1	5116.67	252.18	0	0	10000	5	10252.18	316.89	260	26	2000	1	2576.89	4813
final_2	1824.10	4500	450	8000	4	14324.10	1890.09	2990	299	8000	4	12880.09	1976.16	6910	691	0	0	8886.16	1990.99	2320	232	0	0	4310.99	940
final_3	2966.30	490	49	10000	5	13456.30	2678.29	300	30	2000	1	4978.29	4238.39	20	2	24000	12	28258.39	3090.51	0	0	0	0	3090.51	22
Ratio						2.45						2.58						5.20						1.00	

penalty cost (C_s) and the routing failure penalty cost (C_f) are proportional to the number of spacing violations and the number of buses with routing failures, respectively.

The routing results of the top-3 routers of the contest [2] and ours are summarized in Table 2. According to the contest evaluation metrics, the top-3 routers result in 145%, 158%, and 420% higher total costs than ours. The experimental results show that our router achieves the best performance on minimizing the total spacing violations, which justifies the efficiency and effectiveness of our proposed rip-up and re-route scheme for handling the routing congestion. Note that *the benchmarks provided by the contest organizers are not guaranteed to have solutions without spacing violations*. Moreover, the top-3 routers fail to route some buses while our router only has one routing failure on the benchmark `final_1`. Since `final_1` is a more congested case, the rip-up and re-route stage, which attempts to obtain a violation-free solution, iterates many more times, thus incurring a longer runtime.

In practice, the proposed bus clustering method can merge many buses into super-buses. Therefore, the number of buses is significantly reduced, which reduces the routing complexity. Table 3 gives the numbers of buses without and with bus clustering.

Table 3: Comparison of the numbers of buses w/o and w/ bus clustering.

Circuit	#buses (w/o)	#buses (w/)	Ratio (%)
beta_1	34	4	11.8
beta_2	26	6	23.1
beta_3	60	13	21.7
beta_4	62	14	22.6
beta_5	6	5	83.3
final_1	18	4	22.2
final_2	70	24	34.3
final_3	47	15	31.9

5 CONCLUSIONS

In this paper, we have presented an effective and efficient topology-matching bus routing algorithm considering obstacles and non-uniform track configurations. We have also proposed a bus clustering method that groups buses with two pin groups to reduce the routing complexity. A DAG-based bus routing algorithm has been proposed to efficiently connect all bits within a bus in the specified topology. The rip-up and re-route framework can effectively reduce the number of spacing violations. Experimental results have shown that our algorithm significantly outperforms the top-3 routers in the 2018 CAD Contest at ICCAD. In particular, the first-place router of the contest results in a 145% higher cost than ours.

REFERENCES

[1] *LEMON Graph Library.* www.lemon.cs.elte.hu/trac/lemon, 2014.
[2] *2018 ICCAD CAD Contest.* www.iccad-contest.org/2018/problems, 2018.
[3] *The Boost C++ Libraries.* www.boost.org/users/history/version_1_67_0, 2018.
[4] Y. Badr, A. Torres, and P. Gupta. Mask assignment and synthesis of DSA-MP hybrid lithography for sub-7nm contacts/vias. In *Proc. of DAC*, 2015.
[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition.* 3rd edition, 2009.
[6] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, 1959.
[7] L. McMurchie and C. Ebeling. Pathfinder: A negotiation-based performance-driven router for fpgas. In *Third International ACM Symposium on Field-Programmable Gate Arrays*, 1995.
[8] F. Mo and R. K. Brayton. Semi-detailed bus routing with variation reduction. In *Proc. of ISPD*, 2007.
[9] H.-C. Ou, H.-C. C. Chien, and Y.-W. Chang. Nonuniform multilevel analog routing with matching constraints. *IEEE Tran. on CAD*, 33(12):1942–1954, 2014.
[10] M. M. Ozdal and R. F. Hentschke. Exact route matching algorithms for analog and mixed signal integrated circuits. In *ICCAD - Digest of Technical Papers*, 2009.
[11] M. M. Ozdal and R. F. Hentschke. Maze routing algorithms with exact matching constraints for analog and mixed signal designs. In *Proc. of ICCAD*, 2012.
[12] M. M. Ozdal and M. D. F. Wong. Algorithmic study of single-layer bus routing for high-speed boards. *IEEE Tran. on CAD*, 25(3):490–503, 2006.
[13] M. M. Ozdal and M. D. F. Wong. A length-matching routing algorithm for high-performance printed circuit boards. *IEEE Tran. on CAD*, 25(12):2784–2794, 2006.
[14] J.-T. Yan and Z.-W. Chen. Obstacle-aware length-matching bus routing. In *Proc. of ISPD*, 2011.
[15] T. Yan and M. D. F. Wong. BSG-Route: A length-matching router for general topology. In *Proc. of ICCAD*, 2008.