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

Chen-Hao Hsu\textsuperscript{1}, Shao-Chun Hung\textsuperscript{2}, Hao Chen\textsuperscript{2}, Fan-Keng Sun\textsuperscript{2}, and Yao-Wen Chang\textsuperscript{1,2}

\textsuperscript{1}Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 10617, Taiwan
\textsuperscript{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 the semiconductor industry evolves into the nanometer era, the scale length-matching constraint (e.g., the 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 moves into the nanometer era, the scale length-matching constraint (e.g., the 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.

As the semiconductor industry moves into the nanometer era, the scale length-matching constraint (e.g., the 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.

As the semiconductor industry moves into the nanometer era, the scale length-matching constraint (e.g., the 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.

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 the 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.

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.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure1.png}
\caption{(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 jth pin of the ith bit in a bus, and \( o_i \) denotes the ith obstacle in the design.}
\end{figure}
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, and mask coloring for multiple patterning [4]. To meet the complex rout-

Figure 2 shows an example of the non-uniform track configuration. The bus consists of three 2-pin clusters, and other clusters are overlapped as shown in Figure 2. Any wire routed through the overlap region should have a width satisfying the width constraint of the overlap region because it has a larger width constraint.

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 clusters, and other clusters are overlapped as shown in Figure 3(a), all bits are considered to have the same topology. In Figure 3(b), the bus in Figure 3(b) 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 the bus 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 $\sum_{i=1}^{n} w_i < \sum_{i=1}^{n'} w_i'$. 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 $S_B = \{B_1, \ldots, B_n\}$, and a set of $m$ obstacles $S_O = \{o_1, \ldots, o_m\}$, 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
### Design Rules

- **Routed Routing**: Used 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 CCL. 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 CCL 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 4:** (a) An example of a well-compacted bus. (b) A routing with a higher compactness cost.

(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.

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 $p_{1, a}^{KL}$, $p_{2, a}^{KL}$, $p_{2, b}^{KL}$, $p_{2, c}^{KL}$, $p_{2, c}^{KL}$. The notation $p_{i, k}^{KL}$ denotes the pins of the $i^{th}$ bit in the bus $B_k$; for example, in Figure 6(a), $p_{1, 1}^{KL}$ contains $p_{1, 1}$ and $p_{1, 2}$ in the bus $B_a$. Since all the bits of $B_a$ and $B_b$ in the LCS, $B_a$ and $B_b$ are thus clustered into the super-bus $B_a$ 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 $B_a$ is generated by merging buses $B_a$ and $B_b$. Here, $p_{i, k}^{j}$ 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

$$C_B = \alpha_g \cdot n_g + \alpha_h \cdot n_h + \alpha_f \cdot n_f,$$

where $\alpha_g$, $\alpha_h$, and $\alpha_f$ are the weights for the number of pins at the grid, horizontal, and vertical levels, respectively.
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 \( B \), \( \eta_f \) is the number of continuous failures in the rip-up phase, and \( \alpha_g, \alpha_b, \) and \( \alpha_f \) are weight constants. In our implementation, \( \alpha_g, \alpha_b, \) and \( \alpha_f \) 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 \( \alpha_g \) 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, \( \eta_f \) 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. \( S_B := \text{RandomChooseBits}(B, k) \); // the set of \( k \) randomly chosen bits from \( B \)
2. \( S_g := \emptyset \); // the set of routing guides of the \( k \) bits in \( S_B \)
3. for each bit \( b \in S_B \)
   4. \( g_b := \text{PathSearch}(b, D) \);
   5. \( S_g := S_g \cup \{ g_b \} \);
5. \( PQ_g := \text{TopologyAnalysis}(S_g) \);
6. while \( PQ_g \) is not empty
   7. extract the top guide \( g_{\text{golden}} \) from \( PQ_g \);
   8. \( \eta_f := 0 \);
   9. while \( \eta_f < n_{\text{max}} \)
      10. \( Decision(g_{\text{golden}}) \);
      11. for each bit \( b \in B \)
         12. \( G := \text{ConstructDAG}(b, g_{\text{golden}}, \ D) \);
         13. if no path from \( s \) to \( t \) in \( G \)
            14. \( \eta_f := \eta_f + 1 \);
            15. \( R_B := \emptyset \);
            16. break;
      17. else
         18. \( R_B := \text{ShortestPath}(G) \);
         19. if \( B \) is successfully connected
            20. return \( R_B \);
23. return \( \emptyset \);

**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 \( \text{A}^* \) 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,

\[
k = \min \left( n^B_{\text{bit}}, \max \left( 10, \min(50, \frac{n^B_{\text{bit}}}{4}) \right) \right),
\]

where \( n^B_{\text{bit}} \) is the number of bits in bus \( B \).

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. Figure 7 shows an example of generating the routing guides of a three-bit bus.

![Figure 7](image)

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 \( n_{\text{first}} \) nearest tracks of the segment \( s \) for the both sides are pushed into the track position list UTL\(_1\). 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 \( n_{\text{non-first}} \) 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, \( n_{\text{first}} \) and \( n_{\text{non-first}} \) are defined as follows:

\[
\begin{align*}
n_{\text{first}} &= \min(100, 6n^B_{\text{bit}}), \quad n_{\text{first}} = \min(100, 2n^B_{\text{bit}}),
\end{align*}
\]

where \( n^B_{\text{bit}} \) is the number of bits in bus \( B \).
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.

For two adjacent segments, say \( s_j \) and \( s_{j+1} \), the intersection points of the perpendicular tracks in \( UTL_j \) and \( UTL_{j+1} \) 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 \( s_1 \), which consists of \( v_{1,1} \). The available tracks of \( s_1 \) and \( s_2 \) form box \( s_2 \), which contains three vertices \( v_{1,1}^2, v_{1,2}^2, \) and \( v_{1,3}^2 \).

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 \( box_1 \) to the points in \( box_{j+1} \) 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 \( d_{e \rightarrow M} \left( d_{M \rightarrow e} \right) \). 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 \( d_{e \rightarrow M} \). 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 \( t_{2,1} \) are indexed 1 and 2, respectively.

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.

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.

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

To evaluate the solution quality, the contest [2] provides a cost function considering the metrics described in Section 2. The total cost \( C_{\text{total}} \) 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 \( C_{\text{wl}} \), the normalized segment count cost \( C_{\text{sc}} \), and the normalized bus width cost \( C_{\text{bw}} \). The spacing violation...
penalty cost ($C_r$) 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 which 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 10111. Since 10011 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.

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


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

<table>
<thead>
<tr>
<th>Circuit</th>
<th>$C_r$</th>
<th>$C_s$</th>
<th>$n_v$</th>
<th>$C_f$</th>
<th>$C_{total}$</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta_1</td>
<td>688.70</td>
<td>280.35</td>
<td>0</td>
<td>0</td>
<td>968.70</td>
<td>341.79</td>
</tr>
<tr>
<td>beta_2</td>
<td>514.65</td>
<td>760.95</td>
<td>0</td>
<td>0</td>
<td>1274.65</td>
<td>484.17</td>
</tr>
<tr>
<td>beta_3</td>
<td>1936.28</td>
<td>2044.11</td>
<td>0</td>
<td>0</td>
<td>3980.39</td>
<td>1998.70</td>
</tr>
<tr>
<td>beta_4</td>
<td>1972.41</td>
<td>2209.87</td>
<td>0</td>
<td>0</td>
<td>4182.28</td>
<td>2024.11</td>
</tr>
<tr>
<td>beta_5</td>
<td>114.99</td>
<td>1844.08</td>
<td>0</td>
<td>0</td>
<td>1959.66</td>
<td>98.16</td>
</tr>
<tr>
<td>final_1</td>
<td>362.57</td>
<td>830.33</td>
<td>0</td>
<td>0</td>
<td>1192.87</td>
<td>252.18</td>
</tr>
<tr>
<td>final_2</td>
<td>1824.10</td>
<td>4500.80</td>
<td>4</td>
<td>14324.10</td>
<td>1690.09</td>
<td>1978.70</td>
</tr>
<tr>
<td>final_3</td>
<td>3265.50</td>
<td>4900.50</td>
<td>5</td>
<td>15456.50</td>
<td>2875.29</td>
<td>4238.39</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Circuit</th>
<th>$C_r$</th>
<th>$C_s$</th>
<th>$n_v$</th>
<th>$C_f$</th>
<th>$C_{total}$</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta_1</td>
<td>688.70</td>
<td>280.35</td>
<td>0</td>
<td>0</td>
<td>968.70</td>
<td>341.79</td>
</tr>
<tr>
<td>beta_2</td>
<td>514.65</td>
<td>760.95</td>
<td>0</td>
<td>0</td>
<td>1274.65</td>
<td>484.17</td>
</tr>
<tr>
<td>beta_3</td>
<td>1936.28</td>
<td>2044.11</td>
<td>0</td>
<td>0</td>
<td>3980.39</td>
<td>1998.70</td>
</tr>
<tr>
<td>beta_4</td>
<td>1972.41</td>
<td>2209.87</td>
<td>0</td>
<td>0</td>
<td>4182.28</td>
<td>2024.11</td>
</tr>
<tr>
<td>beta_5</td>
<td>114.99</td>
<td>1844.08</td>
<td>0</td>
<td>0</td>
<td>1959.66</td>
<td>98.16</td>
</tr>
<tr>
<td>final_1</td>
<td>362.57</td>
<td>830.33</td>
<td>0</td>
<td>0</td>
<td>1192.87</td>
<td>252.18</td>
</tr>
<tr>
<td>final_2</td>
<td>1824.10</td>
<td>4500.80</td>
<td>4</td>
<td>14324.10</td>
<td>1690.09</td>
<td>1978.70</td>
</tr>
<tr>
<td>final_3</td>
<td>3265.50</td>
<td>4900.50</td>
<td>5</td>
<td>15456.50</td>
<td>2875.29</td>
<td>4238.39</td>
</tr>
</tbody>
</table>

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

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