

TANGRAM: A Novel ILP-based On-Track Bus Routing via Placement and Compression of Polygons

Jaekyung Im and Seokhyeong Kang
 Pohang University of Science and Technology, Pohang, Korea
 {jkim97, shkang}@postech.ac.kr

Abstract—Bus routing is an advanced topic of signal routing. Unlikely to the classical routing, the bus routing problem has complex constraints such as topology consistency and channel compactness. Existing bus routing algorithms mostly rely on the iterative maze routing, which is heavily time-consuming and sensitive to net ordering, thereby easily succumbing to suboptimality. To overcome this limitation, we propose a novel bus routing algorithm using placement and compression of routing pattern polygons. Critically, our method does not rely on maze routing, thus highly fast and effective. Experimental results show that the proposed method achieves an average of 1.8% quality improvement over the best known results of ICCAD 2018 contest benchmarks.

I. INTRODUCTION

Bus is a predefined set of bits that transfer signal between identical functional units. As technology advances, the datapath has become more wide and this leads to increase of the number of bus structures in the IP blocks. Therefore, enhancing the quality of bus routing is increasingly important in the modern chip design.

The bus routing has distinct constraints, compared with classical signal routing. Since modern chip design flows exploit many timing optimization techniques, it is highly likely that buffers are inserted to problematic bits. To make this step easier, all the bus bits are encouraged to be routed in an isomorphic topology, so one must give attention to maintaining the same routing topology within a bus. Besides, the ICCAD 2018 CAD contest [1] formulated the bus routing problem in the nonuniform track grid, which has incomplete tracks and different width constraints. Following the formulation, the bus routers should take account of nonuniform track configuration as well as typical design rule violations (e.g. spacing rule violations).

In our knowledge, there are four previous works that solve this on-track bus routing problem. Chen et al. [2] proposed MARCH, which simultaneously routes all bits of each bus using a hierarchical routing scheme. Kim et al. [3] proposed a compactness-aware maze routing algorithm (COMPACT). Though it routes a bus in the bit-by-bit manner, topological penalty and compactness penalty are added in the cost function of its A* search, which forces each bit to follow the topology guide as compact as possible. Hsu et al. [4] suggested a bus clustering technique based on the longest common subsequence algorithm. It first merges a set of buses that consist of two-pin nets in the purpose of reducing the problem complexity. Then, each bit is connected during the subsequent maze routing in the direct acyclic graph (DAG). Since the DAG routing gives high cost to (or does not construct) an edge if it induces any design rule violation, it can reduce the number of violations in the maze routing stage. Zhang et al. [5] proposed a detailed routing algorithm based on a Boolean satisfiability (SAT) formulation. After global routing, the router solves a SAT problem to find a set of legal tracks and proper bit ordering. We summarize these works in the Table I.

These bus routers have made great progress, but we would like to point out that all these routers rely on sequential maze routing. Maze routing, mostly referring to Dijkstra algorithm or A* search, is an pathfinding algorithm in a 2D graph. Although it finds the shortest path in the bit level, optimality is not guaranteed in the bus level.

TABLE I: Previous works on ICCAD 2018 CAD Contest [1].

Bus Router	Key Technique	Need Maze Routing?
MARCH [2]	Hierarchical routing	Yes
COMPACT [3]	Bus-level maze routing	Yes
DAG [4]	Bus clustering	Yes
SAT [5]	Topological SAT	Yes
TANGRAM	Polygon placement	No

This is because a routing pattern can be optimal for one bit, while it may not be repeatable in other bits due to the lack of routing resources. Iterative rip-up and rerouting (RRR) can be one solution, but it lacks global view on entire buses, thus leading to converge on a suboptimal solution. Besides, finding a proper bit ordering is another tricky problem since RRR is highly sensitive to bit ordering.

In this paper, we propose TANGRAM¹, a new paradigm of bus routing that does not rely on sequential maze routing. Instead, TANGRAM regards each bus as a set of “polygons” and tries to place them without overlap via integer linear programming (ILP). If it fails to place all the polygons (i.e. fails to route all the buses), it compresses the placed polygons (routed buses) to reserve routing resources for remaining unplaced polygons (unrouted buses). Our experimental results demonstrate that TANGRAM is the most effective, and even the fastest method compared with existing methods [2]–[5].

The main contributions of this paper are as follows.

- TANGRAM is a novel bus routing algorithm based on polygon placement instead of conventional maze routing.
- TANGRAM breaks the leading score of ICCAD 2018 contest, by achieving an average of 1.8% quality improvement over the best known results even with 18.9% reduction of runtime.
- The source code of TANGRAM is available at our GitHub repo <https://github.com/jkim971201/TANGRAM>.

II. PROBLEM DEFINITION

In this paper, we will follow the problem formulation of ICCAD 2018 contest [1]. The objective and constraints are as below.

- Obj** : Find the shortest and most compact routing for all the buses.
Const 1 : Bits must have the same topology with other bits in a bus.
Const 2 : Any pair of wire segments should keep spacing rule.
Const 3 : Wire width must be smaller than its track width.

We assume all the buses consist of two-pin bits in the same manner of the previous works [2]–[5]. For better understanding, we describe four examples in the Fig 1. Note that red, blue and green metals represent the layer *metal1*, *metal2* and *metal3*, respectively. Fig 1a represents the topological violation. *bit1* has layer sequence {*metal2*,

¹The word TANGRAM is a puzzle that consists of flat polygons. Its objective is to replicate a target shape by placing the polygons without overlap.

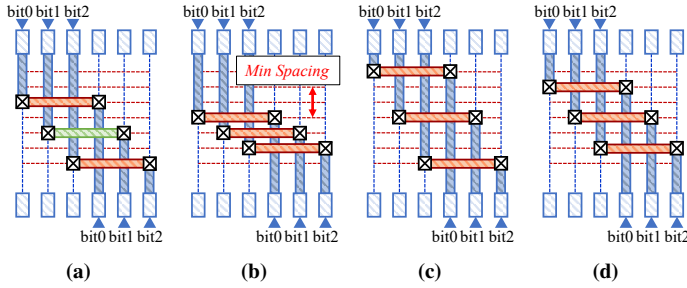


Fig. 1: Four examples of bus routing. (a) Topological violation. (b) Spacing rule violation. (c) Suboptimal bus routing due to non-compactness. (d) Compact bus routing without design rule violation.

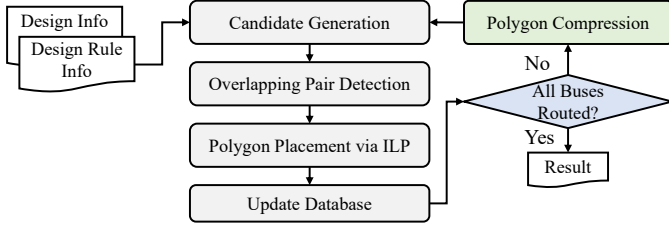


Fig. 2: Overall flow of TANGRAM.

$metal3, metal2\}$, while $bit0$ and $bit2$ have $\{metal2, metal1, metal2\}$. To satisfy the topological constraint (**Const 1**), all the bits must have the same layer sequence, therefore $bit1$ violates the topological constraint. Fig 1b represents the spacing rule violation. If the distance between any two objects – including wires and obstacles – is smaller than the predefined minimum spacing, a spacing violation occurs (**Const 2**). Next, Fig 1c illustrates the non-compact bus routing. The bus routing of Fig 1c does not cause any design rule violation, but it is suboptimal because there exists a more compact bus routing as shown in the Fig 1d. Lastly, any metal wire must be routed on a track and the wire width must not exceed the maximum width specified for each track (**Const 3**). It is worth mention that **Const 2** is a soft constraint in the ICCAD 2018 contest, while other constraints are hard constraints. Details on scoring metric and the penalty with regard to constraint violations will be explained in the Section IV-A.

III. PROPOSED METHOD

Fig 2 illustrates the overall flow of TANGRAM. In the first step, we generate candidate patterns for each bus. Then, we find pairs of candidates that make overlap with each other. To simplify the overlap detection, we extract polygons from the tracks of routing patterns. The overlapping pairs are encoded into constraints of an ILP formulation. Next, we solve the ILP, thereby finding the overlap-free combination of pattern candidates. We commit the routed buses into the database based on R-trees [6], which is a tailored data structure for spatial searching. Similar to COMPACT [3], TANGRAM exploit the R-tree for all the geometrical queries and object management. Meanwhile, if there are remaining buses, we compress the routed buses and repeat these steps until all the buses are routed. If we keep failing to route buses, we also generate multi-bend routing patterns. Details for each step are in the following subsections.

A. Candidate Generation

We should collect a set of routing pattern candidates for polygon placement in the later step. Note that we only collect minimum segment patterns, so every candidates will have two or three segments

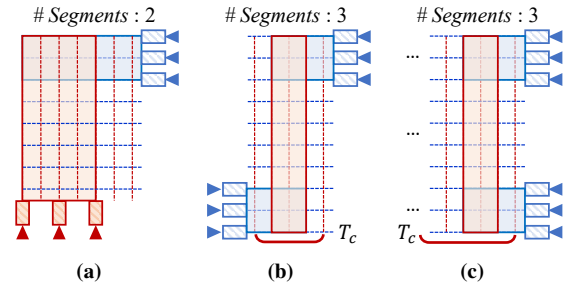


Fig. 3: Examples of minimum segments patterns. (a) A pattern with two segments. (b) A pattern with three segments. (c) A pattern with three segments (when pins are on the same side).

Algorithm 1: Generate Routing Pattern Candidates

Input : Target Bus b , Stride size S
Output: Candidate Patterns C_b

- 1 $C_b \leftarrow \phi, iter \leftarrow 0$
- 2 $\{T_{src}, T_{sink}\} \leftarrow \text{FindPinTracks}(b)$
- 3 $T_c \leftarrow \text{FindCandidateTracks}(b)$
- 4 Sort T_c in descending order
- 5 Track List $T \leftarrow \phi$
- 6 **foreach** track $t \in T_c$ **do**
- 7 **if** t is valid **then**
- 8 Push t to T
- 9 **end**
- 10 **if** $T.size == b.num_bits$ **then**
- 11 **if** T is compact **then**
- 12 **if** $iter \% S == 0$ **then**
- 13 $P \leftarrow \text{ExtractPolygons}(T_{src}, T_{sink}, T)$
- 14 New candidate $C \leftarrow \{T_{src}, T_{sink}, T, P\}$
- 15 Insert C to C_b
- 16 **end**
- 17 $iter \leftarrow iter + 1$
- 18 **end**
- 19 Remove the head of T
- 20 **end**
- 21 **end**
- 22 **return** C_b

as illustrated in the Fig 3. Ignoring the trivial cases of two segments (Fig 3a), the Algorithm 1 and the Fig 4 describe the candidate generation procedure for cases of three segments. We first find the tracks that overlap with either source pins or sink pins (Line 2)². T_{src} and T_{sink} are tracks that will form first and last segment, respectively. Then, we find candidate tracks T_c for the second segment (Line 3). The subprocedure “FindCandidateTracks” returns tracks that locate between the source pins and sink pins. If source pins and sink pins are on the same side, all the tracks between the pin side and the opposite side of the chip are returned (Fig 3c). The candidate tracks T_c are sorted by the descending order of its vertical (horizontal) coordinates (Line 4). The following loop from Line 6 to Line 21 finally finds candidate patterns by traversing the T_c . If it is clear that track t has proper track width and does not make spacing rule violation with abutting tracks in T , then we define the track t is valid and add it to T (Line 7 - 9). When the size of T equals to the number of bits in b ,

²If a pin overlaps with multiple tracks, we carefully select a track that does not make spacing violation (at least that makes minimum spacing violation).

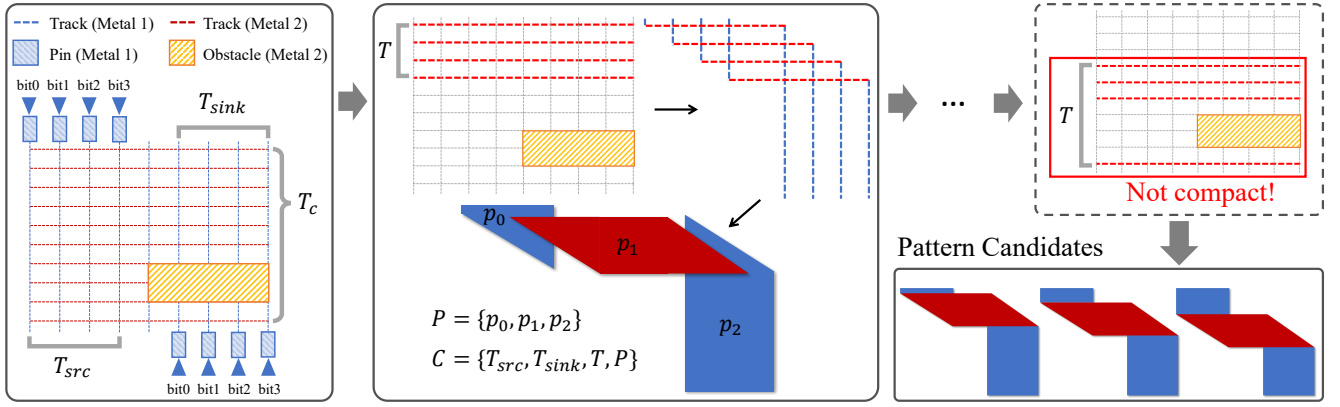


Fig. 4: Workflow for pattern candidate generation. The stride size S is set as 1 in this example.

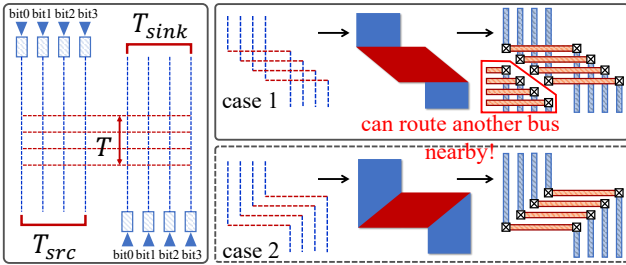


Fig. 5: Two cases of polygon extraction for the three segments pattern. We select the case 1 because it occupies smaller area, so we can route another bus nearby.

we check if T is *compact* (Line 10 - 11). If T turns out to be compact, we extract a set of polygons P (Line 13) and insert a new candidate C to C_b (Line 14 - 15). The subprocedure “ExtractPolygons” is also illustrated in the Fig 4. The blue polygons represent the first segments and the last segments, while the red polygon represents the second segment. One important detail is that we extract a polygon set that occupies minimum routing resources. As Fig 5 describes, there are multiple cases of feasible pattern within the given set of tracks. We select the pattern that occupies less area, so we can pack as many patterns as possible in the future polygon placement.

In the polygon placement process, each pattern candidate becomes a decision variable of ILP. Therefore, it is critical to maintain the number of candidates tractable. To this end, we introduce the stride size S . We generate a pattern candidate only if the iteration number is a multiple of the positive integer S (Line 12). Large S can greatly reduce the problem size, but it is more likely to fail to route all the buses. We empirically find the largest stride size that can succeed to route buses as many as possible in the range from 20 to 35. The effect of stride size will be discussed again in the Section IV-C.

B. Overlapping Pair Detection

To place the polygons of pattern candidates without overlap, we should identify which pairs of candidates make overlap. By regarding a track bundle as a polygon, detection of overlapping candidates reduces to the detection of overlapping polygons. Algorithm 2 describes this procedure and Table II lists the notations. First, we enumerate all candidate pairs from different buses (Line 2). Let P_1 and P_2 are polygons from each candidate C_{pq} and C_{rs} , respectively. Then, we traverse all the polygon pairs (p_1, p_2) , where $p_1 \in P_1$ and

TABLE II: Notations of the ILP Formulation.

Term	Description
B	Set of every buses
C_i	Set of candidates of i th bus
C_{ij}	j th candidate of i th bus
O	Set of overlapping candidate pairs
F	Penalty for unrouted bus
w_{ij}	Weight of j th candidate of i th bus
x_{ij}	Binary variable to denote if j th candidate of i th bus is selected
y_i	Binary variable to denote if i th bus is not routed

$p_2 \in P_2$ (Line 6 - 21). If both p_1 and p_2 stem from either source pin or sink pin, then we do not check overlap (Line 8 - 13)³. This is because there may exist a pair of buses that their bits are interleaved with each other. In this case, the overlap of source/sink pin polygons does not lead to overlap of metal wires, so we do not have to care about these cases. We also do not check overlap when two polygons are from different layers, since it is obvious that this does not make any violation (Line 14 - 16). Lastly, we insert the candidate pair (C_{pq}, C_{rs}) to O if p_1 and p_2 overlap with each other (Line 17 - 19). To circumvent the spacing rule violation, we virtually extend both p_1 and p_2 by half of the minimum spacing.

C. Polygon Placement via ILP

After finding the set of overlapping candidate pairs O , we finally solve the following ILP formulation by using an off-the-shelf solver.

$$\begin{aligned}
 \max \quad & \sum_{i=1}^{|B|} \sum_{j=1}^{|C_i|} w_{ij} \cdot x_{ij} + \sum_{i=1}^{|B|} F \cdot y_i \\
 \text{s.t.} \quad & \sum_{j=1}^{|C_i|} x_{ij} + y_i = 1, \quad i = 1, 2, \dots, |B| \\
 & x_{pq} + x_{rs} \leq 1, \quad \forall \{C_{pq}, C_{rs}\} \in O.
 \end{aligned} \tag{1}$$

We set F as large negative number, so it can work as penalty for routing failure. If i th bus has both source pins and sink pins on the same side (Fig 3c), w_{ij} is defined as wl_{ij}^{-1} , where wl_{ij} denotes the sum of wirelength with regard to candidate pattern C_{ij} . Otherwise, w_{ij} is defined as bit width of the i th bus to maximize the number of routed bits for each iteration. The first constraint is a selection constraint so that each bus has to choose at least one candidate or it

³For a two-pin bit, we always define a pin as a source pin if it is placed higher than the other one.

Algorithm 2: Overlapping Pair Detection

Input : Candidate set for all the buses $\{C_1, \dots, C_{|B|}\}$
Output: Set of overlapping candidate pairs O

- 1 $O \leftarrow \phi$
- 2 $E \leftarrow$ Enumeration of all candidate pairs (C_{pq}, C_{rs}) ($p \neq r$)
- 3 **foreach** pair $(C_{pq}, C_{rs}) \in E$ **do**
- 4 $P_1 \leftarrow$ Polygons of the candidate C_{pq}
- 5 $P_2 \leftarrow$ Polygons of the candidate C_{rs}
- 6 **foreach** polygon $p_1 \in P_1$ **do**
- 7 **foreach** polygon $p_2 \in P_2$ **do**
- 8 **if** both p_1 and p_2 stem from source pins **then**
- 9 continue
- 10 **end**
- 11 **if** both p_1 and p_2 stem from sink pins **then**
- 12 continue
- 13 **end**
- 14 **if** p_1 and p_2 use tracks on different layers **then**
- 15 continue
- 16 **end**
- 17 **if** p_1 and p_2 intersects **then**
- 18 Insert a pair $\{C_{pq}, C_{rs}\}$ to O
- 19 **end**
- 20 **end**
- 21 **end**
- 22 **end**
- 23 **return** O

Algorithm 3: Polygon Compression

Input: Set of candidates C that are selected in the ILP (1)

- 1 $dir \leftarrow$ Determine proper direction to compress
- 2 Sort C by decreasing order of connecting polygon's location
- 3 **foreach** candidate $c \in C$ **do**
- 4 Push the connecting polygon of c as much as possible in the direction of dir
- 5 **end**

can choose to be unrouted. The second constraint prohibits the solver from selecting two overlapping candidates at the same time.

Note that exploitation of ILP for the signal routing already has been widely studied in the literature. Conventional global routers [7]–[9] generate candidates of rectilinear steiner tree for each net on gcell grid, and solve an ILP to route them without capacity overflow. In the bus routing, Streak [10] identifies the possible topologies of each bit and finds the most appropriate solution by solving an ILP formulation. Streak has a cost term for topology variance in its objective and solve the ILP via primal-dual flow. However, we would like to emphasize that the proposed ILP-based polygon placement is much different from the aforementioned methods. We do not find candidates in the bit (net) level, but we rather search for the compact patterns in the bus level. Also, by conceptually substituting segments with polygons, we can efficiently check spacing rule violation instead of exhaustively traversing all the segment tracks.

D. Polygon Compression

If polygons are placed densely where routing resources are in high demand, this may incur routing failure. Fig 6 shows a simple example. One can find that bus A is blocking the sink pins of bus C, so bus C is infeasible to route. To handle this, we can compress the polygons of bus A and B in the upward direction to reserve sufficient space for the

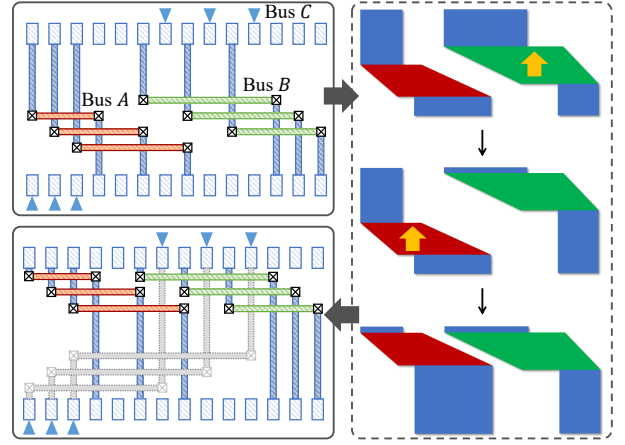


Fig. 6: An example of polygon compression. By compressing polygons of bus A and B, we can reserve sufficient space for an unrouted bus C.

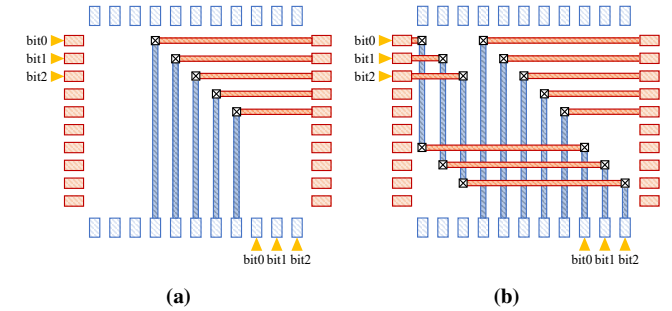


Fig. 7: An example of multi-bend routing. (a) A bus cannot be routed within minimum segments due to the blocking prerouted wires. (b) Three-bend routing pattern can be a solution.

bus C. This process is described in Algorithm 3. We first determine the direction to push polygons (Line 1). If pins are placed like Fig 3c, dir is set as right because it can reduce the wirelength of routed buses. Otherwise, we determine the proper direction, where we can reserve required routing resources for unrouted buses (e.g. dir is set as upward direction in the Fig 6). Next, we sort the candidate set C by decreasing order of the location of connecting (i.e. second) polygon (Line 2). For example, if dir is set as upward direction, a candidate will come first if its connecting polygon is placed most highly. At last, we push the connecting polygon of each candidate as much as possible (Line 4). After pushing the connecting polygon, other polygons within the candidate are lengthened or shortened according to the new location of the connecting polygon. Note that we do not push two segments candidates such as Fig 3a, because they do not have a connecting polygon and all their polygons are fixed.

E. Multi-bend Routing

Even after the polygon compression, there may be still unrouted buses. If iterations of polygon placement keep failing to route the remaining buses, TANGRAM allows non-minimum segments candidate for these buses. We call this technique as “Multi-bend Routing”. To explain this, we depict an example in Fig 7. Since the weight of a pattern candidate is set as bit width of each bus, a bus of small bit width may not be routed during polygon placement (Fig 7a). In this case, even the polygon compression cannot make this bus routable because two segments buses are not compressed during

the procedure. To route these troublesome buses, we exceptionally generate candidates of multi-bend pattern (non-minimum segments) during the candidate generation. Additionally, if a bus does not have compact and spacing violation-free candidates, we also allow non-compactness and spacing rule violation for the bus in this step. Thus, TANGRAM can search for more various patterns and finally can route all the remaining buses via the polygon placement (Fig 7b). The multi-bend candidate generation is only an extension of the Algorithm 1, so we will omit its detailed description due to page limit.

IV. EXPERIMENTAL RESULTS

TANGRAM is coded with approximately 7k lines of C++ including Boost library [11] and the ILP solver in the Google OR-Tools [12]. We use ICCAD 2018 contest benchmarks [1] and compare the results with four previous works [2]–[5]. The specifications of each testcase are listed in the Table III. Our experiments are executed on a Linux machine equipped with Intel Xeon Gold CPU 2.3 GHz. For the fair comparison, we use four threads which the contest have specified as maximum number. Also, the official evaluator provided by the contest is used for score evaluation.

A. Evaluation Metrics

The quality of bus routing is evaluated by the scoring metric C_{total} defined by the ICCAD 2018 contest [1].

$$C_{total} = C_{route} + C_{space} + C_{fail}, \quad (2)$$

where C_{route} , C_{space} , C_{fail} represent the sum of routing cost, sum of violation penalty and sum of routing fail penalty, respectively. The routing cost C_{route} is defined by following equation.

$$C_{route} = \alpha \cdot \sum_{bus\ b} C_{wl}^b + \beta \cdot \sum_{bus\ b} C_{seg}^b + \gamma \cdot \sum_{bus\ b} C_{com}^b, \quad (3)$$

where C_{wl}^b , C_{seg}^b and C_{com}^b are cost for wirelength, segments and compactness of bus b , respectively. α, β, γ are weighting parameters for corresponding cost terms. $C_{wl}^b, C_{seg}^b, C_{com}^b$ are defined as

$$\begin{aligned} C_{wl}^b &= \sum_{bit \in b} \frac{WL(bit)/MinWL(bit)}{NumBits(b)} \\ C_{seg}^b &= \frac{NumSegs(b)}{NumMinSegs(b)} \\ C_{com}^b &= \sum_{seg \in b} \frac{Width(seg)/MinWidth(seg)}{NumSegs(b)}. \end{aligned} \quad (4)$$

The minimum wirelength (MinWL) is computed by half perimeter wirelength and the minimum number of segments (NumMinSegs) is computed as shown in the Fig 3. The spacing violation cost C_{space} is defined as

$$C_{space} = \delta \cdot N_{space}, \quad (5)$$

where δ is a penalty weight for spacing rule violation and N_{space} represents the number of spacing rule violations. The routing failure cost C_{fail} is defined by following equation.

$$C_{fail} = \epsilon \cdot N_{fail}, \quad (6)$$

where ϵ is a penalty weight for routing failure and N_{fail} represents the number of buses that fail to be routed.

B. Comparative Study

We compare both the routing quality (C_{total}) and runtime of TANGRAM with other bus routers in Table IV. The detailed results

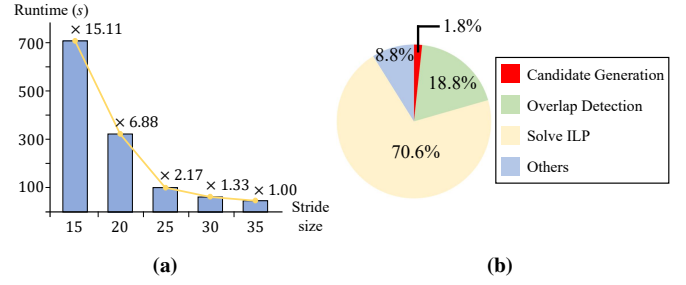


Fig. 8: (a) Relationship between the stride size and runtime (measured in beta_1). (b) Runtime breakdown (combined every testcases).

TABLE III: ICCAD 2018 Contest benchmarks information.

Benchmark	Metric Weights					Design Information				
	α	β	γ	δ	ϵ	#Bus	#Bit	#Track	#Layer	#Obs
beta_1	5	1	5	8	2000	34	1260	49209	3	159
beta_2	5	1	5	8	2000	26	1260	49209	3	0
beta_3	12	1	4	8	2000	60	665	22732	3	555108
beta_4	12	1	4	8	2000	62	698	22702	3	0
beta_5	8	1	5	8	2000	6	1964	54150	4	0
final_1	10	1	5	10	2000	18	1032	81226	3	0
final_2	10	1	5	10	2000	70	1285	14209	3	0
final_3	10	1	5	10	2000	47	852	21379	4	0

C_{wl} , C_{seg} and C_{com} are also reported. In the SAT [5] paper, the authors reported the results of two different versions – “before CNC” and “after CNC”. We cite the results of “after CNC” because it makes the better routing quality. TANGRAM achieved about 1.85% improvement of routing quality on the best known results, which were made from DAG [4]. Though TANGRAM loses for DAG [4] and SAT [5] in final_2 and final_3, respectively, it is clearly superior for the rest. We remark that the testcase beta_5 has too dense pin placement, so it is impossible to route beta_5 without spacing rule violation or routing failure.

On the other hand, TANGRAM is also the fastest router among all the other routers. It showed about 18.9% improvement of runtime, compared with the second fastest one, MARCH [2]. We ascribe this improvement to the proposed techniques that liberate the router from exhaustive iterations of time-consuming maze routing and RRR.

C. Runtime Analysis

To show the effect of the stride size S on runtime, we sweep S from 15 to 35 in the testcase beta_1. The results are presented in Fig 8a, where the numbers along yellow line are runtime normalized by that of when $S = 35$. One can see that even 15.11 \times larger runtime is taken when $S = 15$, compared to that of when $S = 35$. This is because the large number of pattern candidates make the ILP formulation (1) more untractable. However, we have observed that too large stride size can lead to routing failure. Therefore, we select the proper S between four numbers {20, 25, 30, 35} for each testcase.

The runtime breakdown result is given in Fig 8b, which shows that “Solve ILP” is the runtime bottleneck of TANGRAM. To make this plot, we combine the runtime of each step for every testcases. “Others” includes the time spent on database update and polygon compression. The polygon compression takes only trivial portion (< 1%), so we merge its runtime to “Others”.

V. CONCLUSION

In this paper, we have presented a new bus router, TANGRAM. Unlikely to conventional bus routers, TANGRAM does not perform maze routing. Instead, it iteratively places and compresses the routing pattern polygons. Experimental results show that TANGRAM clearly

TABLE IV: Comparison on the ICCAD 2018 Contest benchmarks. C_{wl} , C_{seg} and C_{com} are scaled with α , β and γ , respectively. † : Since the previous works [2]–[5] are not publicly available, we cite the score and runtime results of DAG from its paper [4], while others are cited from SAT paper [5]. We would like to note that there are no great differences in CPU spec between ours and cited papers [4], [5].

	Benchmark	Routing Cost				Penalty Cost		C_{total}	Runtime† (s)
		C_{wl}	C_{seg}	C_{com}	C_{route}	C_{space}	C_{fail}		
TANGRAM (Proposed)	beta_1	33.8	34.0	95.9	682.6	0	0	682.6	49
	beta_2	25.9	26.0	72.1	515.8	0	0	515.8	51
	beta_3	69.1	60.0	250.7	1891.8	0	0	1891.8	113
	beta_4	72.0	62.0	303.0	2138.1	0	0	2138.1	75
	beta_5	6.0	5.0	13.1	118.5	400	0	518.5	5
	final_1	18.6	21.4	26.6	340.0	0	0	340.0	144
	final_2	70.8	74.0	238.5	1974.3	120	0	2094.3	246
	final_3	46.9	51.3	556.2	3301.4	0	0	3301.4	46
	Normed Average	100.00	100.00	100.00	100.00	100.00	-	100.00	100.00
SAT [5]	beta_1	33.8	34.0	96.0	683.0	0	0	683.0	11
	beta_2	25.9	26.0	72.2	516.2	0	0	516.2	6
	beta_3	69.2	60.0	249.7	1923.7	0	0	1923.7	187
	beta_4	72.5	62.0	302.4	2172.7	0	0	2172.7	72
	beta_5	6.0	5.0	13.1	118.5	400	0	518.5	288
	final_1	18.5	24.0	26.7	342.3	0	0	342.3	206
	final_2	73.3	74.0	236.6	1989.7	0	0	1989.7	533
	final_3	52.5	49.3	614.9	3648.9	0	0	3648.9	237
	Normed Average	102.50	100.19	103.57	103.95	76.92	-	102.72	211.54
DAG [4]	beta_1	33.9	34.0	96.5	685.8	0	0	685.8	2
	beta_2	25.9	26.0	72.8	519.4	0	0	519.4	26
	beta_3	72.2	60.0	250.7	1929.8	0	0	1929.8	71
	beta_4	75.0	62.0	303.6	2176.9	0	0	2176.9	3
	beta_5	6.0	5.7	13.4	120.7	600	0	720.7	103
	final_1	18.3	21.4	29.4	351.9	0	0	351.9	1920
	final_2	69.5	74.7	243.4	1987.6	290	0	2277.6	548
	final_3	47.0	56.6	501.2	3032.2	0	0	3032.2	88
	Normed Average	101.39	101.98	97.11	98.56	171.15	-	101.85	379.26
COMPACT [3]	beta_1	34.1	36.0	94.9	680.4	0	0	680.4	330
	beta_2	26.0	26.7	72.6	519.3	0	0	519.3	199
	beta_3	70.1	60.7	250.8	1905.6	0	0	1905.6	80
	beta_4	73.1	63.3	302.1	2148.9	0	0	2148.9	59
	beta_5	5.0	4.0	10.2	94.9	0	2000	2094.9	9
	final_1	18.6	24.0	27.0	345.0	0	0	345.0	951
	final_2	71.7	79.4	234.3	1966.3	240	0	2206.3	676
	final_3	48.7	56.6	549.9	3293.2	0	0	3293.2	38
	Normed Average	101.23	105.10	99.09	99.92	46.15	-	114.90	321.52
MARCH [2]	beta_1	33.9	34.0	112.4	765.1	0	0	765.1	50
	beta_2	25.9	28.7	84.6	580.8	0	0	580.8	9
	beta_3	72.4	62.0	252.7	1942.5	0	0	1942.5	72
	beta_4	76.5	71.4	293.4	2165.4	0	0	2165.4	39
	beta_5	6.0	5.7	12.6	118.5	1848	0	1966.5	12
	final_1	18.4	22.0	30.1	355.8	840	0	1195.8	352
	final_2	69.6	81.3	258.8	2071.0	1480	0	3551.0	199
	final_3	47.0	51.3	558.4	3312.6	150	0	3462.6	133
	Normed Average	101.89	106.80	103.01	103.19	830.38	-	136.12	118.96

outperforms existing bus routers, in the aspects of both solution quality and runtime. We believe the polygon placement and compression are of interest and amenable to other problems in VLSI signal routing.

ACKNOWLEDGEMENT

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2023-00222609, Development of thermal modelling and heat-spreading architecture for heterogeneous integrating package of AI chiplets).

REFERENCES

- [1] *ICCAD 2018 Contest*, <https://www.iccad-contest.org/2018>
- [2] J. Chen, J. Liu, G. Chen et al., “MARCH:MAze Routing Under a Concurrent and Hierarchical Scheme for Buses”, *Proc. DAC*, 2019.
- [3] D. Kim, S. Do, S.-Y. Lee et al., “Compact Topology-Aware Bus Routing for Design Regularity”, *IEEE TCAD* 39(8) 2020, pp. 1744-1749.
- [4] C.-H. Hsu, S.-C. Hung, H. Chen et al., “A DAG-based Algorithm for Obstacle-Aware Topology-Matching On-Track Bus Routing”, *IEEE TCAD* 40(3) 2021, pp. 533-546.
- [5] H.-T. Zhang, M. Fujita, C.-K. Cheng et al., “SAT-Based On-Track Bus Routing”, *IEEE TCAD* 40(4) 2021, pp. 735-747.
- [6] A. Guttman, “R-trees: A Dynamic Index Structure for Spatial Searching”, *Proc. SIGMOD*, 1984.
- [7] J. Hu, A. Roy and I. L. Markov, “Sidewinder – A Scalable ILP-Based Router”, *Proc. SLIP*, 2008.
- [8] M. Cho and D. Z. Pan, “BoxRouter: A New Global Router Based on Box Expansion and Progressive ILP”, *IEEE TCAD* 26(12) 2007, pp. 2130-2143.
- [9] T.-H. Wu, A. Davoodi and J. T. Linderth, “GRIP: Global Routing via Integer Programming”, *IEEE TCAD* 30(1) 2011, pp. 72-84.
- [10] D. Liu, B. Yu, V. Livramento et al., “Synergistic Topology Generation and Route Synthesis for On-Chip Performance-Critical Signal Groups”, *IEEE TCAD* 38(6) 2019, pp. 1147-1160.
- [11] *Boost C++ Library*, <https://www.boost.org>.
- [12] *Google OR-Tools*, <https://github.com/google/or-tools>.