Obstacle-Aware On-Track Bus Routing

Vitor H. M. Pereira, Marcus H. S. Mendes, José A. M. Nacif
[1]Science and Technology Institute
Universidade Federal de Viçosa, Florestal, Brazil
{vitor.h.pereira, marcus.mendes, jnacif}@ufv.br

***Abstract*—The great advances achieved in very large scale integrated technology cause an increase in the difficulty for routing the buses through the spacing while minimizing the cost. The term bus defines a communication system that connects internal components in an integrated circuit. The routing process consists in placing the wires from the bus on space. There are some rules, related to wire width, the spacing between obstacles, routing direction, and others, we need to respect. The routing process can be separated in global routing and detailed routing. This paper presents a solution passing in both processes. In the global routing, we use a differential evolution based approach in a graph representation of the area to find a brute path of minimum length to guide the following step. By the next, we use a greedy selection to place the wires on the tracks. We compare the results achieved using a classic flux algorithm and three different mutation operators in differential evolution in global routing.**

***Index Terms*—VLSI, IC, routing, bus, differential evolution**

I. Introduction

The current complexity of the Very Large Scale Integrated (VLSI) technology transforms the bus routing is a very challenging task. Some optimizations aiming to improve the overall functionality of these systems impose constraints on the interconnection of elements [4]. Those factors, like the spacing between elements, necessity of pass between small obstacles maintaining the topology, use of multiples layers with non-uniform track configuration, etc., are important to consider when routing the buses. The main goal of routing is to find out a sequence of tracks and vias to place the wires that connect two or more components from the integrated circuit.

This class of problem divides the routing area into different layers, where each one has a routing direction (the tracks inside is either vertical or horizontal) and a set of obstacles. Each layer can only be connected with your adjacent layers, through a via. Generally, these are two steps in traditional routing: 1) global routing: where the routing area is divided into tiles then turned into a graph that represents possible paths; and 2) detailed routing, focus on place wires on available tracks.

The routing process, besides be crucial in IC's development, has a high level of difficult: it is necessary a great number of resources to complete this task respecting all constraints and with a good cost. To illustrate their importance, this problem was one of the three at ICCAD CAD Contest 2018 [1], a challenging research and development annual competition focused in solve real issues from industry on Electronic Design Automation field. It gathers competitors all over the world to work in challenges provided by industrial companies, fostering industry-academia collaborations. Even although that provide solutions from others competitors, we do not use as a comparison in this work, because they are not publicly available yet.

This work presents an algorithm that analyzes the structure of the routing area in order to route the more number of buses possible, keeping the topology, without crashing between two or more of them. It can be seen that are many different forms to work with each constraint that can be used to represent the problem (such as the number of layers, grid and/or direction fixing). A mix of the best strategies on each one is the challenging key of this work.

In the remaining of this paper, we first show the creation of graph representation of the routing area, followed by global routing creating. Detailed routing comes after, and lastly the results are shown.

II. Related Work

There many approaches in bus routing that solves problems with some differences. For example, [3] presents a solution in gridless layers, while [10] focus with one layer problems.

The routing process can be split into global and detailed routing, in order to limit the search space [6]. Besides detailed routing takes majority part of execution, the most improves are made in global routing.

We use some insights from those approaches cited before in our solution. Between them, [10] proposes the creation of independent regions, that illustrates obstacle-free areas in routing space, by slicing space with obstacles boundaries and then merging adjacent ones. [3] and [2] presents a connection graph approach addressed to find a possible route in gridless representations. According to [6] gridded approaches use a Dijkstra's algorithm variants to get shortest paths in one layer situations. Besides, [7] presents a heuristic approach, using Differential Evolution to find a better solution. Although all of them works on bus routing, the constraint on those solutions are slightly different, distinguishing them of our problem.

III. Problem Formulation

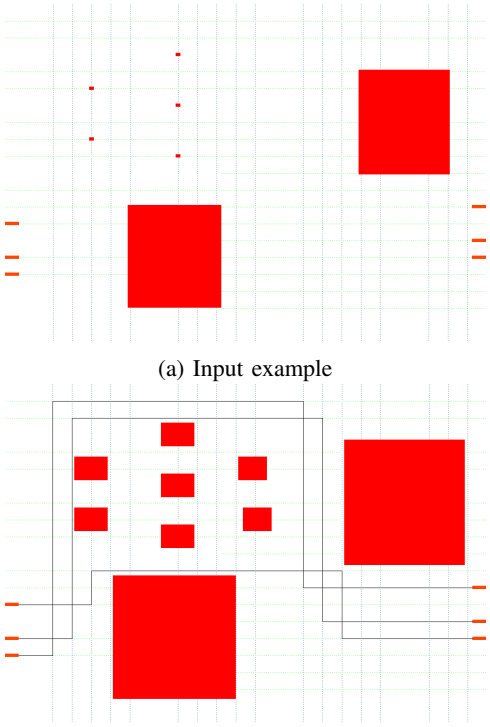
This problem consists in, given a routing area and a set of obstacles, connect the pins of a group of buses using the available tracks on different layers. Both routing area and obstacles are defined by rectangles. Each layer contains:

- a routing direction, that defines if all tracks inside it are horizontal or vertical;
- a spacing needs to be held between every item;

- his set of tracks, that each one is defined by a line and their width.

Lastly, a bus is defined by a set of bits with two or more pins, that have a width constraint for each layer.

The Figure 1 presents a example of routing. In this figure, the orange rectangles represents the pins of the bus, the red rectangles are obstacles and the dot lines are tracks. The continuous lines in 1b represents the wires placed after the routing.



(a) Input example

(b) Output example

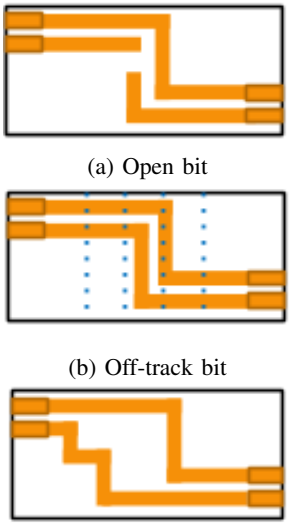
Fig. 1: Example of an input (a) and your respective output (b). All layers are drawn together for simplicity purposes.

The overall cost (Equation 8) calculate the quality of the solution: the smaller it is, better the solution. It consists of the result from routing cost with a penalty applied to it. There are two types of penalty: spacing and route fail. The first consists in a spacing constraint not respected, and sum a weighted cost according to Equation 5. Otherwise, a routing fail disqualifies the solution, for this bus. The fail penalize by Equation 6. There are three factors that can define a routing failure, illustrated by Figure 2, that is:

(a) A bit is disconnected
(b) A wire is off any track or in a track with lesser width capacities
(c) A bit not follow the topology. In other words, all bits from a bus need to have the same number of segments with the same sequencing.

If any of this happens, the routing is considered fail.

The routing cost is evaluated due three measures: 1) wire length cost, that consist in a normalized sum of length of all



(a) Open bit

(b) Off-track bit

(c) Topology error

Fig. 2: Example of routing fails.

wires, calculated as shown in Equation 1; 2) segment cost, that count the number of segments, described by Equation 2; and 3) compactness cost, that calculate how close is the segments, given by Equation 3. In an ideal case, all of those values are close to 1. The constants associated of each measure on Equation 4(α, β and γ) are given on the input, and can change for each problem. The cost functions are described below:

$$C_{wi} = \frac{\sum_{j=1}^{All\ bits\ of\ bus\ j} \frac{wire\ length\ of\ bit\ j}{half\ parameter\ wire\ length\ of\ bit\ j}}{\#bits\ of\ bus\ i} \quad (1)$$

$$C_{si} = \frac{\#segments\ of\ bus\ i}{lower\ bound\ \#segments\ of\ bus\ i} \quad (2)$$

$$C_{ci} = \frac{\sum_{j=1}^{All\ segments\ of\ bus\ i} \frac{width\ of\ segment\ j}{lower\ bound\ width\ of\ segment\ j}}{\#segments\ of\ bus\ i} \quad (3)$$

$$C_R = \sum_{i}^{All\ buses} \alpha * C_{wi} + \beta * C_{si} + \gamma * C_{ci} \quad (4)$$

$$P_s = \#spacing\ violations * \delta \quad (5)$$

$$P_f = \#failed\ buses * \epsilon \quad (6)$$

$$C_P = P_s + P_f \quad (7)$$

$$Overall\ cost = C_R + C_P \quad (8)$$

IV. Methodology

For routing, we first order the buses according to the number of wires. As the cost of routing fail is the same for any bus and the bigger occupy more resources, it is worth route the minors first. We find the graph representation of the area initially, once it is the same for all buses. Then, for each bus, we perform global followed by detailed routing. The Figure 3 presents the flux of our algorithm.

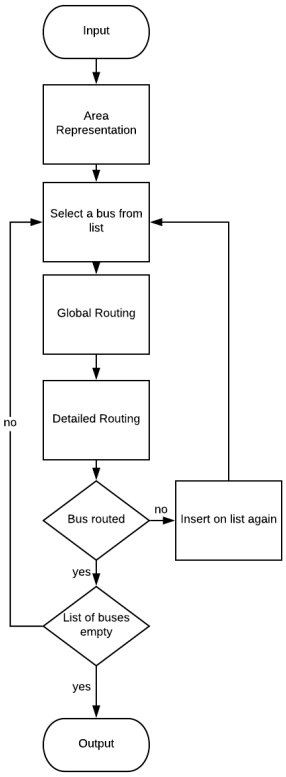


Fig. 3: Routing Flow

A. Creating the Graph

For global routing, we use a weighted graph for represent the routing areas. Each vertex denotes a free area (where are no obstacles inside) and the edges are adjacency between areas, with their weight equal the number of tracks on these connections. The use of this representation ensures that any segment placed in one of his areas will not collide with obstacles.

To mount this graph, we use the steps ahead (exemplified by Figure 4):

1) Expand obstacles: besides avoid collides, there is a need to keep a spacing between obstacles and wires. To include this on our representation, we increase the size of obstacles equal to spacing constraints, so the spacing becomes an interdicted area. In the Figure 4, from 5c to 4b has an increasing of the obstacles size, representing this process.
2) Define free areas: to get the space division, we create lines along the obstacles borders and routing area limits. Those lines define rectangles where is guaranteed to have no obstacles inside. These rectangles can be used as vertices. The merging is according the routing direction of the layer: if is an horizontal, first we merge the rectangles horizontally and then vertically, otherwise the opposite is done.
3) Merge areas: a large number of items are created in the last step, that could lead to a high execution time. So we merge the areas that have an exactly common border, transforming small areas into a bigger without cover any obstacle.
4) Getting edges: basically, there are two types of edges: the ones in the same layer and that who connect two layers. The first is defined by the existence of a border between the rectangles, while the second occurs when two areas in different layers overlap each other. In both cases, the weight is the count of tracks on those encounters.

We first executed the steps 1 to 3 in each one. How these steps use only data from one layer, in other words, there are no dependencies between layers, we can process them in parallel, gaining performance. However step 4 connect adjacent layers, so it is only performed after the previous be finished on all layers.

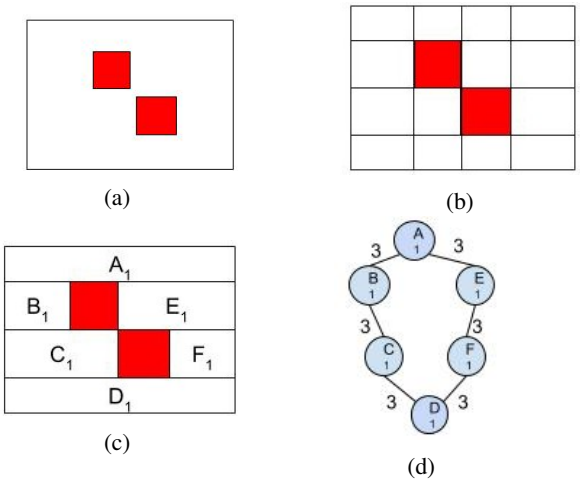


Fig. 4: Example of graph creation. (a) is the original layer, (b) presents the obstacles expanding and free area creations, (c) demonstrates the merging and (d) is the final graph for this layer.

B. Global Routing

Once we have the graph representation, we can perform the global routing, in order to indicate the areas we can get tracks to connect the pins. This step consists in finding a Steiner Tree. The best solution are indicated by an Minimum Rectilinear Steiner Three. However this problem is well-known as an NP-complete [5]. The MRST is a tree that connects all terminals with the minimum cost.

Initially we mark the terminals node, those vertices who contains pins. Each bus has, at least, two sets of pins. Because of the way we create the graph, the entire set not necessary will be on the same node. However, we need only one source and one sink. For help this, we create two new nodes on the graph: one to be the source connected in original input nodes and other to be the sink connecting all outputs. The Figure 5 presents this process.

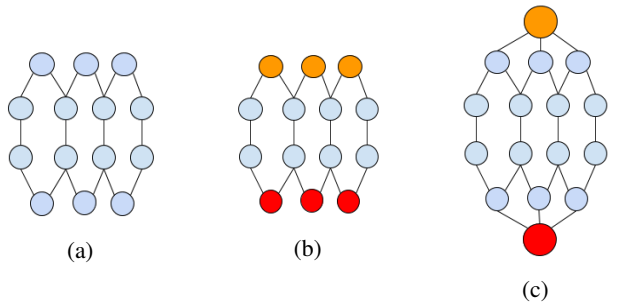


Fig. 5: Example of terminals union on graph. (a) is the original graph. (b) shows the marked terminals and (c) the union of them

An early solution to find an Steiner Tree is the use of a flux algorithm. On the given graph is used an implementation of the Ford-Fulkerson method, defining a target flux, as the number of wires from the bus, and getting one path that fulfills this flux. This algorithm is a greedy solution for the problem, once it gets the first route that passes all wires.

Trying to improve the solution, we can use a differential evolution approach. As Differential Evolution is used in continuous space [9], we use a modified version proposed by [7].

The DE technique consists in the generation of representations of solutions, called individuals, passing through an initialization followed by a sequence of mutation, crossover, and selection towards finding a global optimum. Each pass is explained below.

- Initialization: consist in a generation of a population of *M* individuals. An individual is an adjacency matrix representation from a graph. Each one starts with an initial solution created by randomly selecting edges, and their weights, from the original graph. We ensure the feasibility from those individuals, in other words, they represent a feasible graph connecting the initial and final nodes. Besides find the MRST is hard, find a non-optimal solution is easier.
- Mutation: a process that transforms an individual, switching some cells values according to a random factor, since new solution remains feasible. An individual is considered feasible if: 1) the edges weights not pass the number of wires from the bus; and 2) the topology is not broken for any wire. The classic mutation is represented by Equation 9. The variables v and x are vectors usually, but they are a matrix in our solution, as we use this representation. $p1, p2, p3 \in [1, M]$ are random numbers, $i \neq p1 \neq p2 \neq p3$ and F is a number. Furthermore, originally F is a real number, but as we are in discrete space we use as an integer.

$$v_i = x_{p1} + F(x_{p2} - x_{p3}) \tag{9}$$

Other types of mutations that can be used also, as exemplified in [8]. The Equation 10, denoted as DE/rand/2, is an example. It uses 2 scaling factors F, giving more spread solutions.

$$v_{i,G} = x_{p1} + F_1(x_{p2} - x_{p3}) + F_2(x_{p4} - x_{p5}) \tag{10}$$

This last Formula (11) presents a other method for mutation. In this formula, $x_{i,G}$ is the trial individual (we explain it in the next step).

$$v_i = x_i + F_1(x_{p1} - x_i) + F_2(x_{p2} - x_{p3}) \tag{11}$$

- Crossover: combines one new selected individual, named trial, with the mutated one, by switching their columns controlled by a crossover ratio. We generate a random number: if it is greater than a given crossover ratio we copy the column from the trial matrix, otherwise we copy from the mutant. The new individual created in this process is named target solution.
- Selection: gets the best between target and trial individuals. Their fitness, that shows how good is the solution, is calculated by counting segments used: The less used, better. If the target individual has minor fitness compared with the trial, it included in next generation.

The Figure 6 presents the differential evolution flux of execution. These steps are repeated until the stop criteria are reached. In this work, we use a simple maximum of generations equal 10000 as criteria.

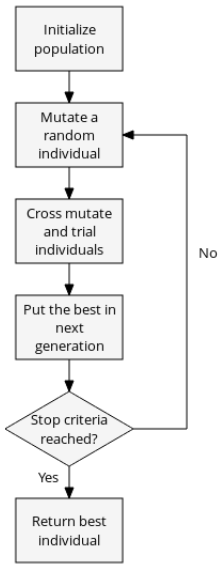


Fig. 6: Flowchart from Differential Evolution Algorithm

To maintain the topology on all bus, we need to pass every wire in nearly the same route, so the whole bus is routed as a unity. One bus is routed at a time, but as the problem requires route multiples buses, there is a need to refresh the graph, removing from the use edges the capacity already spent.

C. Detailed Routing

This final step is the more delicate process to be solved: we need to select on which tracks we will place each wire, using the areas found by global routing.

We define a queue with the initial nodes (those that belong to the first set of pins). To set the firsts active tracks, to guide the path to the next node, we get one track that overlaps with your respective pin (according to the problem formulation, there at least one track overlapping with each pin). If this track is in the same layer from pin we can pass for next step directly, but it can be on another layer. In this case, we need to create vias connecting those layers before proceeding.

For every node, starting from the initials cited before, we add all his children on a queue, if it is not a final node (has no pins), and do the internal routing based on actual and his next nodes. To maintain the same topology on all bus, we route all wires inside a node before pass for the next, so we need to save the active track (last used for placing) for each wire. In the internal routing we have two possibilities: the next node is 1) in the same layer or 2) an adjacent layer.

In the first case, we need to connect the active track with a going out. If it can not be done directly, we find the first one that meets the requirements of width and connect those two by a free segment on another layer. That segment is first sought out in adjacent layers, and deeper if necessary, creating vias to connect those layers. Once we can get an out track, we set this the active track and pass for the next node.

If the next node is in an adjacent layer we just find a possible track, according to requirements, inside the next node and get the intersection point. A via is also created and we advance, refreshing the active track.

How two wires can not cross, every segment created add an extra obstacle. This creates a need for always test if has a collision before creating the wire, once this obstacle is not represented in the graph created previously.

This process is repeated while the queue is not empty. After this, is made the routing inside the final nodes. They have a different approach. In this case, we don't seek an out track, but the pin of his wire. So we get one track that overlaps with the pin, what is guaranteed to exist, and get the connection with the active track as we did before.

The Figure 7 presents an example of detailed routing: 7a shows the nodes selected by global routing, with tracks and the pins in orange; 7b presents the initial active tracks set, in red. In this case a direct route can be traced for the next node, so the destiny tracks are equal to the actives ones; 7c demonstrates the creation of segments on the first node and passing for next. The actives tracks is set to be the destiny tracks, in this case they still the same; 7d show the creation of segments from second node. In this case, we use a middle segment on another layer for connecting origin an destiny tracks.

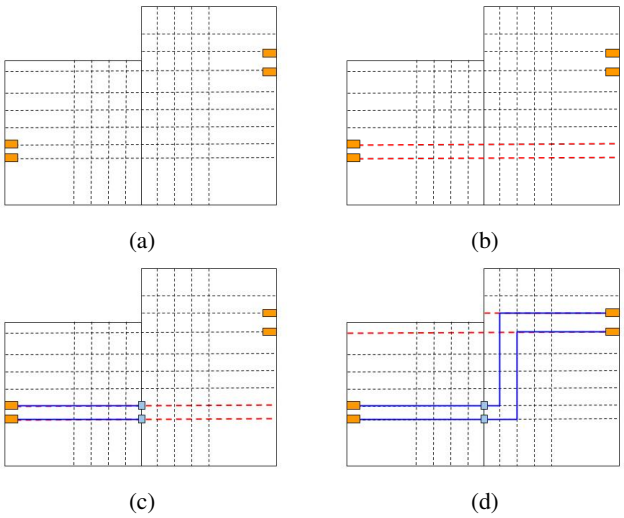


Fig. 7: Example from detailed routing. Only one layer with all resources is shown for simplicity purposes.

If a bus can not be fully connected with this previous steps all segments are discarded, avoiding use spaces that could connect another bus. After try route all buses, another try is done for those not connected. As a great number of routing resources is likely to be used, there is a large probability of another route be selected on global routing.

V. Results

Our algorithm was developed using the C++ language and executed in an Intel i7 4770 with 16GB of memory. The initial part of the process, that defines the free areas for each layer, is built in parallel since they did not affect each other. For test were used 8 cases, described in Table I, consisting in parts of real designs simplified. Those designs were provided by the ICCAD CAD Contest 2018, as part of their contest. As explained before, we not use the others contestant results for comparison once there is no base for this: their works are not publicly available yet.

	#nets	#buses	#tracks
beta_1	1260	34	49209
beta_2	1262	26	49209
beta_3	665	60	22732
beta_4	698	62	22732
beta_5	1964	6	54150
final_1	1032	18	81226
final_2	1285	70	14209
final_3	852	47	21379

TABLE I: Test cases configurations

In Table II besides total cost, we show a detailed cost achieved, presenting the routing cost, spacing penalty and routing fail calculated as shown in problem formulation, respectively the Equations 4, 5 and 6.

	CR	Ps	Pf	Total Cost
beta_1	3252.18	6128	6000	15380.18
beta_2	2564.26	5984	2000	10548.26
beta_3	7428.14	8246	8000	23674.14
beta_4	7274.38	7528	14000	28802.38
beta_5	4342.62	4584	0	8926.62
final_1	4264.14	6730	2000	12994.14
final_2	8212.44	9728	16000	33940.44
final_3	4362.82	4580	4000	12942.82

TABLE II: Detailed results with our flux algorithm

This result shows that the only case had every bus routed. All others keep, at least, one unrouted bus. Also, there is a considerable number of spacing penalties, the most of them between wires or vias from another bus. It shows a point of improvement on detailed routing.

To evaluate the impact on global routing, in Table III we present the routing cost achieved, calculated by Equation 8, with 4 different algorithms:

1) Ford-Fulkerson, the flux method based algorithm;
2) DE/rand/1, a differential evolution with the classic mutation (random individuals selected and one scaling factor);
3) DE/rand/2, also a differential evolution with random individuals selected on mutation, but with two scaling factors;
4) DE/cur-to-rand/1, other differential evolution algorithm, but with mutation using a combination of the trial individual with randoms selected.

Those differential evolution methods only discern on mutation, each one using an operator. Their operators are explained on Global Routing section, and are guided by equations 9, 10 and 11. For evaluation were used populations of 1000 individuals, crossover ratio of 0.8 and the following values: $F = 1$, $F_1 = 2$ and $F_2 = 1$.

	Ford-Fulkerson	DE/rand/1	DE/rand/2	DE/cur-to-rand/1
beta_1	15380.18	12244.18	**9482**.18	10142.18
beta_2	10548.26	8288.26	8102.26	**5864.26**
beta_3	23674.14	21648.14	**16820.14**	18048.14
beta_4	28802.38	24680.38	21204.38	**16928.38**
beta_5	8926.62	8842.62	5068.62	**4898.62**
final_1	12994.14	10684.14	8120.14	**6860.14**
final_2	33940.44	31826.44	**24962.44**	27206.44
final_3	12942.82	10896.82	**5824.82**	8004.82

TABLE III: MRST algorithms comparison

The graph from Figure 8 shows the sum of costs from all test cases for each variation. We can see that the flux algorithm has the bigger cost, showing their inefficiency compared with others. The two variations from the classic mutation on DE algorithm, DE/rand/2 and DE/cur-to-rand/1, has similar results and are the best. Comparing execution time, the flux one is about 2x faster than DE's. The variation of mutation does not affect so much the time.

VI. Conclusion

This work create a multilayer gridded routing method working under a strict set of constraints. Initially, we transform the routing space in a graph representation to simplify the routing. We explore a set of different approaches in global routing, finding the best suitable between them. The improving on algorithm was focused on global routing, opening possibilities to studies on detailed routing. Finally, we use a simple placement wire in detailed routing, performing it as a batch, ensuring the topology maintenance.

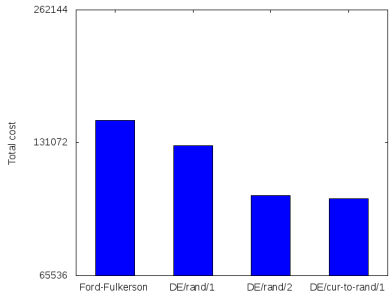


Fig. 8: Sum of costs from all test cases

Our algorithm could route most of buses from simplifications of real designs from industry of VLSI. The results shows best solutions using the differential evolution algorithm. As it was developed for continuous problems, the use of other evolutionary strategies for discrete problems (like Tabu Search or Iterated Local Search) can improve the solution. The detailing shows a considerable spacing penalty, it creates space for improving on detailed routing, avoid this cost.

References

[1] *2018 CAD Contest*, 2018. *Available at http://iccad-contest.org/2018/.*

[2] Jason Cong, Jie Fang, and Kei-Yong Khoo. An implicit connection graph maze routing algorithm for eco routing. In *Computer-Aided Design, 1999. Digest of Technical Papers. 1999 IEEE/ACM International Conference on*, pages 163–167. IEEE, 1999.

[3] Jason Cong, Jie Fang, and Kei-Yong Khoo. Dune-a multilayer gridless routing system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(5):633–647, 2001.

[4] Jason Cong, Lei He, Cheng-Kok Koh, and Patrick H Madden. Performance optimization of vlsi interconnect layout. *Integration, the VLSI journal*, 21(1-2):1–94, 1996.

[5] Michael R Garey and David S. Johnson. The rectilinear steiner tree problem is np-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

[6] Michael Gester, Dirk Müller, Tim Nieberg, Christian Panten, Christian Schulte, and Jens Vygen. Bonnroute: Algorithms and data structures for fast and good vlsi routing. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 18(2):32, 2013.

[7] Suvrajit Manna, Tyajodeep Chakrabarti, Udit Sharma, and Subir Kumar Sarkar. Efficient vlsi routing optimization employing discrete differential evolution technique. In *Recent Trends in Information Systems (ReTIS), 2015 IEEE 2nd International Conference on*, pages 461–464. IEEE, 2015.

[8] Karol Opara and Jarosław Arabas. Comparison of mutation strategies in differential evolution–a probabilistic perspective. *Swarm and Evolutionary Computation*, 39:53–69, 2018.

[9] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[10] Jin-Tai Yan and Zhi-Wei Chen. Obstacle-aware length-matching bus routing. In *Proceedings of the 2011 international symposium on Physical design*, pages 61–68. ACM, 2011.