Semiconductor Optoelectronics
ISSN: 1001-5868

ROBUST RECTILINEAR STEINER TREE CONSTRUCTION: A MULTI OBJECTIVE OPTIMIZATION USING GENETIC ALGORITHM

Chittaranjan Mohapatra[1, 2] and Nibedita Adhikari[1]

[1]Utkal University, Vani Vihar, Bhubaneswar, Odisha, India
[2]Silicon Institute of Technology, Patia, Bhubaneswar, Odisha, India
nibedita.cs@utkaluniversity.ac.in

Abstract. The genetic algorithm is used to generate an optimal rectilinear Steiner tree with no obstacle overlap. To accelerate the algorithm, for the initial solution, Delaunay Triangulation is used to generate an adjacency list of neighbour nodes in less time. A multiple objective optimization technique is developed to obtain a Robust Rectilinear Steiner Tree (RRST).The primary purpose is to achieve sufficient reduction in wire-length by connecting two pins without any obstacles. Some pins are difficult to reach due to major obstacles and become outliers. The second objective is to add fewer non-pins in order to connect the outliers. A heuristic is used to link the outlier pins while minimising the count of non-pins. The third research goal is to decide uopon the minimum possible Steiner points by using common path. For its cutting-edge nature, this method can be very well applied to (Very Large-Scale Integration) VLSI physical design routing.

Keywords: Rectilinear Steiner Tree, VLSI, Routing, Delaunay Triangulation, Obstacle, Genetic Algorithm, Optimization.

1 Introduction

One of the key elements that can significantly help with the optimization of VLSI circuit layout is routing, which is susceptible to losses of energy and higher routing cost. The use of Rectilinear Steiner Minimal Tree in VLSI routing was crucial in constructing the rectilinear route for power and clock data. The available literature has solely been focused with obstacle avoidance and the minimum wire-length achievable. However, with fewer Steiner points, a few possible non-pins can further reduce wire length of an Obstacle Avoidance Rectilinear Steiner Minimal Tree. Such a tree may have some detour paths that increasethe wire length. The present study's major focus is to build a Robust Rectilinear Steiner Tree (RRST) applying a multi-objective optimization approachthat produces least possible wire-length, non-pins and Steiner points, with zero overlap.

The paper is organized as follows: the Section 2 gives a review of the history and related works that have been finished so far. In the Section 3, RRST building methods were proposed. The results and comparative analysis arediscussed in theSection 4. The conclusion statements are presented in Section 5.

2 Background and Literature Survey

In any VLSI circuit design the pins are connected in a rectilinear path in order to consume less power and minimum space. This rectilinear path is termed rectilinear Steiner tree. The research on the circuit design reveals that the Steiner tree should be a minimal one. The following section describes Steiner concepts and various existing research works on it.

2.1 Rectilinear Steiner Minimal Tree (RSMT)

If P is a collection of n points, locate a set S of Steiner points that minimizes a rectilinear path over $P \cup S = \{p_1, p_2, \dots, p_n, s_1, s_2, s_3, \dots, s_l\}$. An $RSMT$ edge cost is the distance between its ends, and $RSMT's$ cost is the total of its edge costs in Manhattan distance. In a VLSI layout, the wiring cost between two pins p_i and p_j is typically modelled as an edge e_{ij}which is calculated based on the $x$and y coordinates of the pin in the layout, as shown in Equation 1.

$$Cost(e_{ij}) = \|x_i - x_j\| + \|y_i - y_j\| \tag{1}$$

Every edge in an $RSMT$ is a route between the two end points made up of one or more alternating horizontal and vertical segments.

A Minimum Spanning Tree (MST) and RSMT are portrayed by Robins and Zelkovsky for the same point set in the Manhattan plane as seen in the Fig. 1[27].

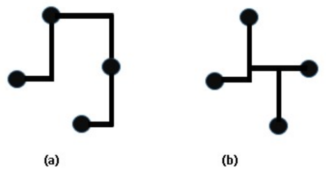

Figure 1: (a) MST (b) RSMT where solid dots are pins in VLSI Circuit

2.2 Related Work

There are various ways to solve a Steiner Tree problem, including branch and bound solutions, dynamic programming-based methods [24, 11]. A powerful search strategy of Steiner point may be effective for solving various Steiner Tree problem. A swap vertex method for neighborhood search is established by Fu et al. to improve quality of an initial MST [3]. The method exchanges a vertex with another and reconstruct the MST. A heuristic is designed to avoid recurrence of similar moves and to focus the searching to suitable search areas. However, the complexity of the method makes it less affordable.

To enhance the optimal assurance, an advanced Steiner point selection is developed by Liu et al [17]. After the initial solution, the line segment of a 2D U-formed pattern is replaced with a group of line segments for maximum connected component to determine the optimum Steiner points. Some of the work has been done on obstacle avoidance Steiner tree. The layout of the gates and macros areas, can be a connection hurdles. However, it has no effect on the interconnection path between pins. An Obstacle aware non-Manhattan distance-based routing algorithm is proposed by Ghosal et al [4]. This type of routing is called X Architecture based routing which lowers the count of Steiner points which results in enhance chip performance. The algorithm can be improved if obstacle aware process can be applied at the time of construction of the MST.

Lin et al. constructed an obstacle avoidance spanning tree by transforming the slant edge to rectilinear edges and introduced Steiner points [15]. This spanning tree was polished in three stages: the overlap edge was removed first, followed by the redundant node, and lastly the U-shaped pattern was refined. They claimed a maze routing based obstacle avoidance rectilinear Steiner tree [16]. This method was divided into three stages: selecting Steiner points, ripping up Steiner points, and rebuilding Steiner points. To obtain an optimal collection of Steiner points, the second and third steps were repeated.

Next Pajor et al. proposed a Guarded Multi-start algorithm that was implemented on a branch and bound routine [24]. This was a three-pass approach, the first pass used breadth-first search (BFS) to find a set of Graph Cut and its arcs. The other passes traversed the set and perform an augmentation by reducing the residual capacity of each arc. Nath et al. employed Gradient Descent and Particle Swarm Optimization in a mixed meta-heuristic optimization method [22]. In a circular sequence, PSO served as the algorithm's global search component, and gradient descent serves as the algorithm's local search component. Even though it requires time to finish, it always discovered the best solution. Kundu et al. included a Pseudo Boolean satisfy-ability (PB-SAT) model for producing a rectilinear Steiner tree that met all of the specified PB-SAT requirements [14].

Guo et al. proposed a Divide-and-conquer strategy based method obstacle avoidance Steiner tree where it divided the initial solution to specific number of vertices and merge them with Physarum-inspired routing algorithm to get the refined solution [5]. Huang et al. generated an initial solution whose information were stored in a Lookup table [8]. Then it generated an Obstacle avoidance Steiner tree by considering the rectangles corner points which was refined using a heuristics method.

Guo et al. also presented a GPU-based RSMT version of the FLUTE method for large-scale datasets [6]. The decomposition procedure employed a parallel breadth-first search (BFS) algorithm. A net was partitioned into two sub-nets at an edge, whose end points acted as a common pin. This edge was used as a bridge to combine solutions and retain the relations of sub-nets within the net. Another GPU version based on the KMB algorithm tried to make this technique more efficient by utilizing CUDA [21].

Steiner Tree problems are solved using heuristic algorithms based on the concept of combining the Dijkstra and Prim's algorithms to determine the shortest route by Tran et al. [28]. Zhang et al. proposed a methodology for iteratively reducing any detour connections by improving the pre-solution Steiner point candidate until the quality is optimum [30]. An example of a detour connections to an optimal connection is shown in the Fig. 2(b)–(d). Hu et al. proposed a heuristic for construction of rectilinear Steiner tree based Ant Colony Optimization. However it was further enhanced while considering obstacles [7].

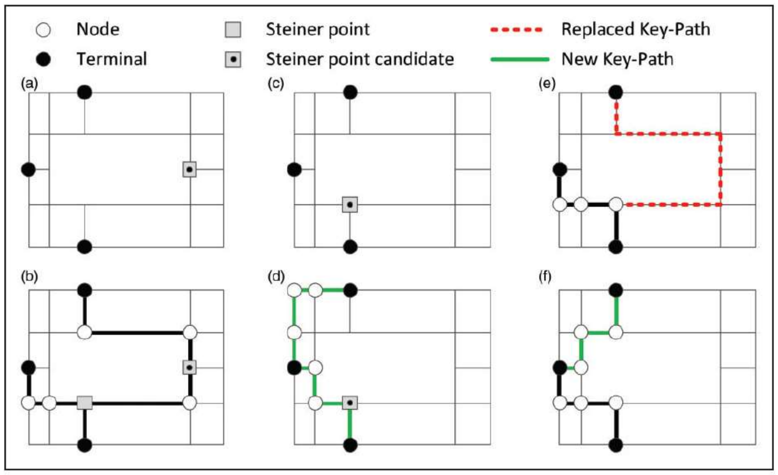

Figure 2: Detour Path Elimination

To build an obstacle avoidance Steiner tree, a particle swarm algorithm and a union find data structure were used by Liu et al. [18]. Joshi and Thakur devised an on-chip routing method combining the Genetic algorithm and the Cuckoo search algorithm [13]. Both pieces of work can be used to build the Manhattan Steiner tree. Zavoianu et al. approached an evolutionary-based optimization paradigm as multi-objective evolutionary algorithms [29]. This work can be further enhanced for hybridization with a discretization of the design surface.

There are few studies performed on the minimum possible Steiner points. In case of an obstacle avoidance Steiner tree, it is still a huge demand to connect outlier nets including some non-pins. A study on the inclusion of reduced number of non-pins is also another challenge in obstacle avoidance Steiner tree. A Steiner tree with less number of non-pins and Steiner point results in an optimal wire-length which can be of utmost importance in VLSI physical design.

3 Proposed Method

This section describes the suggested RRST approach which is created in three stages. By avoiding obstructions, the first phase provides an initial solution that generates a rectilinear Steiner tree of the pins. This step may fail to build the optimal Steiner tree due to significant barriers in the path between certain pins known as outliers. The second step finds these outliers and adds some non-pins to create a plausible path to the outliers. The final RRST is created in the third stage using a genetic algorithm. The mechanism of these three stages are described in detail below.

3.1 Rectilinear Steiner Tree Construction

For any set of points a Minimum Spanning Tree can be produced in less time if Delaunay Triangulation is used instead of complete graph [20]. At first, an edge-list is generated using Delaunay Triangulation graph. These edge-list are inserted into a Min Priority Queue. Then Find-Union disjoint data structures are used to generate the RMST by using rectilinear distance. The idea is to select an edge with minimum cost, provided that edge is not facing any obstacles. When one edge is popped out the Min Priority Queue, it checks, whether it faces any obstacles or not. If a Steiner point between two vertices, falls on an obstacle then it is discarded.

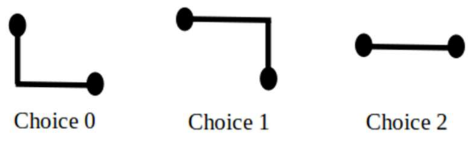

Figure 3: Choice between two Points

In this study, it is assumed that there are three possible choices for a Steiner point, as illustrated in the Fig. 3. If option 1 falls within an obstacle zone, option 2 is used. If both options land in an obstacle region, the path is deleted. If the pins are linear and no Steiner point is required, the third option is used. If any of the options is available, it is determined whether the path passes through any obstacles. If the edge does not come into contact with any other path or impediment, the path is clear and the pins are linked.

Algorithm 1: Obstacle Avoidance Rectilinear Steiner Tree

```
Input: Set of Pins $P$ and Obstacles $O$
Output: Obstacle Avoidance Rectilinear Steiner Tree
1.  $E = Delaunay\ Triangulation\ (P)$
2.  $Create\ a\ Grid\ A = zeros(x_{max}, y_{max})$
3.  $Set\ A[i][j] = \infty\ where\ i,j \in O$
4.  $for\ v \in P\ do$
5.      $Makeset(v)$
6.  $end\ for$
7.  $Q = MinPriorityQueue(E)$
8.  Cost=0
9.  $while\ Q\ do$
10.     $e(u,v) = Delete(Q)$
11.     $if\ Find(u)\ != Find(v)\ then$
12.         $s = Find\ Steiner\ Point(u,v)$
13.         $if\ s \in A\ where\ A \neq 0\ then$
14.             $continue$
15.         $end\ if$
16.         $if\ e(u,v)\ intersect\ any\ path\ and\ s \notin A\ where\ A > andA < \infty\ then$
17.             $continue$
18.         $end\ if$
19.         $if\ e(u,v) \in A\ where\ A == \infty\ then$
```

```
20.                 $continue$
21.             $end\ if$
22.             $Union(u, v)$
23.             $Cost = Cost + Cost_{e(u,v)}$
24.         $end\ if$
25.     $end\ while$
26.     $return\ Tree$
```

A grid is crucial in the obstacle avoidance procedure because it keeps track of the path and barriers. The usage of a grid lowers the time complexity of path search to $O(1)$. Its proof is provided in Theorem 1. The grid is a maximum-sized matrix comprising the x and y coordinates of the provided set of pins. Initially, all values are set to zero, and the obstacles' locations are filled with ∞. If an edge's route consists entirely of zeros, then it is available. If it is non-zero and not ∞ and falls linearly over another, then it is also made available. If the path consists of ∞, then the path faces obstacle and discarded. If a path is available and its end points belong to separate sets, then a union operation is performed. The total cost is calculated when each edge is added to the tree during union operation. If a path repeats, it considers once only. Finally, this method returns the Obstacle Avoidance Rectilinear Steiner Tree that includes the cost of the tree, its nodes and list of edges. The Algorithm 1 returns this rectilinear Steiner tree of all pins by avoiding Obstacles as described.

Theorem 1: *If P is a set of n pins in a chip with m obstacles, and then all pins can be connected in a rectilinear path in $O(n\ log\ n)$ time without intersecting any obstacle.*

Proof: A set of edges E can be generated by applying Delaunay Triangulation to get the relationship among n pins. According to Raza [26], any triangulation of P has $(2n - 2 - k)$ triangles and $(3n - 3 - k)$ edges if all pins are in a plane and k pins are on the boundary of the convex hull of P. This yields a linear number of edges, i.e. $|E| \approx O(n)$.

A min-heap based priority queue is used to store the list E. It returns the minimum distance edge in $O(n\ log\ n)$ times [2]. A union and find disjoint data structures is used to connect all pins. The number of executions of the dis-joint data structures depends on the number of edges in the priority queue [2]. That is $O(E) \approx O(n)$ time.

A grid matrix with the largest value of all pins x and y coordinates is constructed, and all values are set to zero. This takes $\theta(1)$ time. The values of grid are set to ∞ in the areas of obstacles. For m obstacles this will take $O(m)$ times. If no path intersects an obstacle, two pins are connected. This intersection test involved checking the content of a few columns in a certain row. If j columns in a row are to be tested, this can take $O(j)$ times. So, the total time complexity of Rectilinear Steiner Tree Construction in Algorithm1 is $O(n \log n)$ times.

3.2 Outlier Detection and Connection

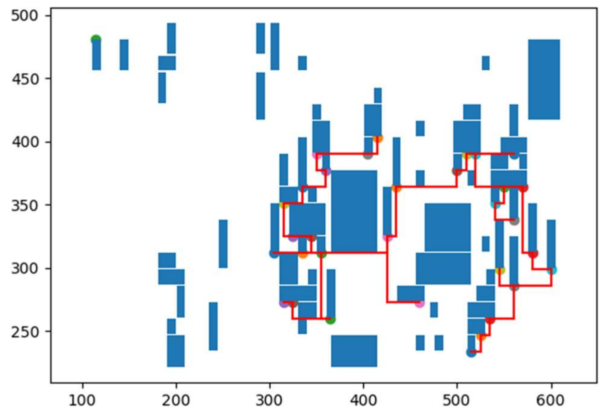

Figure 4: Outlier Pin at x=115 y=481 in IND5

The outlier pins are the pins that do not have a direct path to their nearest pin dueto Obstacles. The Fig. 4 shows an outlier in IND5 benchmark instance. A heuristics is designed to detect outliers and connect them with its nearest pin. It lowers the execution time of the RRST, the wire length and the number of Steiner points. The procedure of the heuristic is described in Algorithm2. To detect outliers, an Obstacle Avoidance Rectilinear Steiner Tree must first be constructed. Therefore, this tree is the input to the Algorithm 2.

The first step is to find the disconnected components. Then, a random leaf pin v is chosen whose degree is either 1or 0. Another nearest pin u is calculated from remaining leaf pin of other component sets. The leaf pins v and u are supposed to be the border locations of two linked components, and the candidate pin to bridge two connected components. A connected component may have a single node with degree 0, therefore it is also taken into account.

Algorithm 2: Outlier Detection and Connection Algorithm

```
Input: Obstacle Avoidance Rectilinear Steiner Tree
Output: Modified Obstacle Avoidance Rectilinear Steiner Tree
1.    $if\ |connected\ components| > 1\ then$
2.        Get a outlier leaf nodes $v$ where node.degree == 1 or node.degree == 0
3.        Find the nearest leaf nodes $u$ to the outliers
4.        Determine the obstacle across the pin $v$ and $u$
5.        Select the bottom-left and upper right corner points of the obstacle
6.        Create a new set of points$P_{new}$= all pins + the corner points as non-pin
7.    $end\ if$
8.    Use Algorithm 1 with the $P_{new}$
9.    Remove all edges having a leaf non-pin as an endpoint from the tree
10.   return Modified Obstacle Avoidance Rectilinear Steiner Tree
```

The obstacles between v and u are identified and their bottom-left and upper-right corners are added based on a diagonal location of u and v. These two corners are known as non-pins. A new set of points are formed by adding these non-pins to the set of pins. The Algorithm 1 is called again to obtain a modified Obstacle Avoidance Rectilinear Steiner Tree by providing the new set of points as its input.
The result of the IND5 benchmark after adding the non-pins to the set of pins is shown in the Fig. 5. This strategy ensures that a solution in the initial population is a connected component while also introducing some redundant edges R_1 and R_2 as shown in Fig. 5. These edges have a leaf non-pin as an end point. So these edges can be easily identified and removed because they are not required to connect two pins.

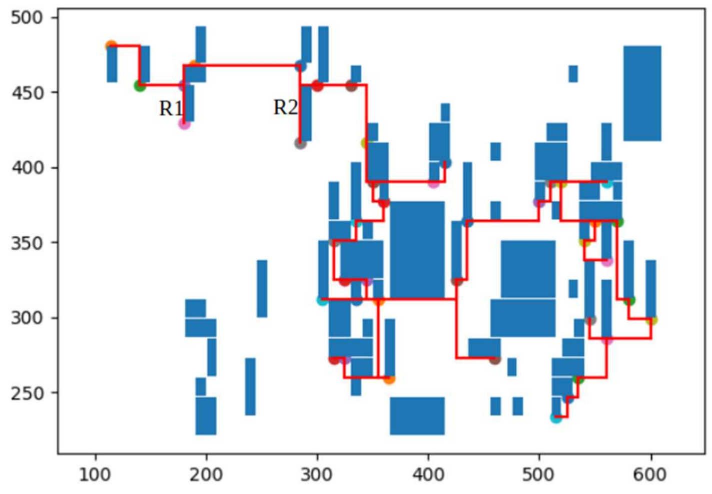

Figure 5: Connected Outlier Pin of IND5 Benchmark Data

3.3 RRST

The proposed RRST is an obstacle avoidance Steiner tree the produces an optimal wire-length considering various situation. This section defines the objective function and a genetic algorithms that fits Steiner tree problems. The algorithm finally output a connected optimal Steiner tree without obstacles.

Problem Formulation

The Obstacle Avoidance Rectilinear Steiner Tree may have some detour paths, as the choice of path is system determined when it comes across an obstacle. These path can be enhanced further with some optimization kind of algorithm. In this section a genetic algorithm based optimization mechanism is used to arrive at an optimal wire-length of the tree.
Let E be the set of possible edges and O be the set of obstacles $\{o_1, o_2, \ldots, o_m\}$, then the optimization task to construct minimum $cost(T)$ having a minimal path can be defined in equation 2.

$$\min cost\ (T) = \sum_{e_{ik}, e_{kj} \in E} e_{ik} + e_{kj} \qquad (2)$$

Subject to

$\forall s_k \notin O,$

$e_{ik}, e_{kj} \cap O = \phi,$

$|S| < n + m * m$

In Equation 2, s_k is a Steiner point constructed to link two pins p_i and p_j, while e_{ik} and e_{kj} are the horizontal and vertical edges between the two pins. The main objective is to minimize the wire-length as $cost(T)$ such that no Steiner point falls inside obstacle, none of the edges $e \in E$ intersects any obstacles and the total number of Steiner points created, as well as the number of non-pins included to connect some outliers, should be kept as low as possible. Furthermore, the essential aspect of the optimization is analyzing the recurring path only once, allowing the overall Wire-length to be reduced.

The Genetic Algorithm

The concepts of natural evolution are the foundation of the stochastic search method known as the genetic algorithm (GA). Individuals in a community fight with one another for survival in nature as the population evolves. Greater fitness allows offspring's to live, while lesser fitness results in death. In GA, fitting chromosomes (solutions) are more likely to take part in genetic operators and pass on their genes to the next generation. Genetic operators aid GA in both exploring new areas of the search space and utilizing the most hopeful regions of the search space. Following is a discussion of the suggested GA structure, including Encoding, Initial Population Generation, Fitness Function, Selection, Crossover, Mutation, Evaluation, and lastly Updation.

Encoding: A Steiner tree is represented as a chromosome (solution) of the proposed GA. An edge-set encoding method [25, 23] is used to represent a solution. This encoding offers high locality, is heritable and is adaptive to problem-specific genetic operators. An edge is augmented with weight, associated points, their Steiner point, and Steiner choice. All edges are assumed to be obstacle-free edges because each solution is an Obstacle Avoidance Rectilinear Steiner Tree.

Generation of initial solutions: All feasible Obstacle Avoidance Rectilinear Steiner Trees are generated by applying random Steiner point choice between two points in Algorithm 1. The population set T has s solutions.

Fitness Function: The fitness function computes the total cost of the tree with less number of Steiner points without any obstacles. This computation is incorporated with the generation of each solution. This is same as the objective function in equation2.

Selection: The selection is a crucial process in a GA. The binary tournament selection method and roulette wheel selection are used in this phase. First, two solutions are selected randomly from the population, then a tournament is performed, and the winner is selected for the mating pool. Similarly, half of the population is generated with a binary tournament solution, and the rest is generated using the roulette wheel selection method. The mating pool is updated in the Initial Population.

Crossover: The crossover rate is considered as 1.0, 0.9 and 0.8 for sensitivity analysis of the results obtained from GA. A heuristic crossover technique is used in the proposed algorithm. Two trees are selected randomly and input for the cross over operation. The common edge between these two trees are added to the child tree by using the union-find set data structure. The remaining edges of both trees are chosen at random and appended to the new spanning tree using the union-find set data structure. A clear picture of crossover operation is shown in the Fig. 6. Here E_1, E_2, E_3, E_4, E_5, E_6 are the edges of the two parent trees which are not common. Out of theses six edges, three edges E_1, E_3, E_4 are selected to generate a child tree. Here union-find disjoint data structure is used construct the tree. During the child tree generation the total cost and number of Steiner points are calculated. This child tree may be a solutions that can survive for next generation.

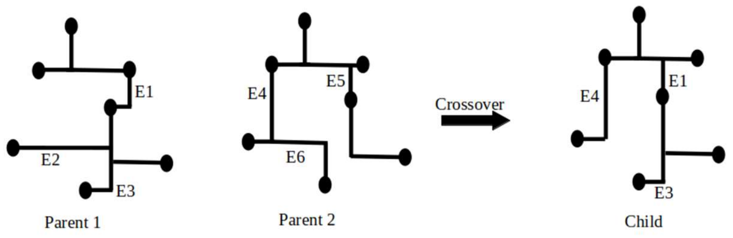

Figure 6: Crossover Operation of two Chromosomes

***Mutation**:* In Mutation operator edge transformation technique is applied. To ensure that the spanning tree has been connected, an arbitrarily selected spanning tree solution is chosen, and an edge is deleted and replaced with another randomly selected edge. In the Fig. 7 the mutation operation is explained in details. One edge E_1 is randomly selected in a random solution as parent and using union-find data structure strategy a child spanning tree is constructed by adding edge E_2 to it.

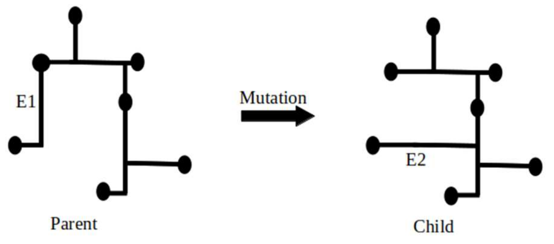

Figure 7: Mutation Operation of a Random Chromosome

Evaluation and Updation: The best solution is selected from the initial population prior to the start of evolution. The total cost and the number of Steiner points of a new solution are

compared with the best solution. If the new one has a better solution, then the population is updated with the new one by replacing the tree with the largest cost.
The complete GA framework of the proposed approach is mentioned in Algorithm 3. The Crossover and Mutation operation pseudo-codes are specified in Algorithm 4 and Algorithm 5 respectively.

In Algorithm 3, T1, $T_2, \ldots, T_s$ are chromosomes of the population with size s and T_b stores the minimum cost among the solutions of the population. A mating pool is created using the binary tournament and roulette wheel selection method. Then the population is updated with the mating pool. The crossover operation is applied between two randomly selected solutions as parents, P_1 and P_2, to get a solution T^C as child. If the child solution has less cost and fewer Steiner points, then the best solution is replaced with the maximum cost spanning tree in the population. Similarly another solution is selected randomly and the mutation operator is applied to it, and it returns a child solution T^C. Again, the population is updated with the new solution.

In each generation, the best solution is compared against the child solution. If the child's solution is better than the best solution, the best solution is replaced, and the population is updated. The probability parameter of selection, crossover and mutation is determined empirically. The Algorithm 3 terminates after certain number of generations. The time complexity of the algorithm is determined below and stated as Theorem 2 followed by the proof.

Algorithm 3: Optimization Algorithm

```
Input: Modified Obstacle Avoidance Rectilinear Steiner Tree
Output: RRST
1.      $Set, Population\ Size\ as\ s$
2.      $Generate\ initial\ population\ T = T_1, T_2, ., T_s\ using\ Algorithm\ 1$
3.      $T_b = Best\ of\ T$
4.      $Set, count = 0\ and\ the\ number\ of\ generation\ as\ g.$
5.      $while\ count < g\ do$
6.          $Perform, Binary\ tournament\ selection$
7.          $Perform, Roulette\ wheel\ selection$
8.          $Update\ T$
9.          $P_1 = Selection(T)$
10.         $P_2 = Selection(T)$
11.         $T^c = Crossover(P_1, P_2)$
12.         $if T^c < T_b\ then$
13.             $T_b = T^c$
14.             $Update\ T$
15.         $end\ if$
16.         $p = Selection(T)$
17.         $T^c = Mutation(p)$
18.         $if\ T^c < T_b\ then$
```

```
19.             $T_b = T^c$
20.             $Update\ T$
21.         $end\ if$
22.     $end\ while$
```

Algorithm 4: Crossover Process

Input: Particle P, Particle Q
Output: New Particle

```
1.  $Common\ Edges = Intersection(P, Q)$
2.  $Difference\ Edges = Union(P, Q) - set1$
3.  $A\ new\ particle\ P_{new} = Union\ Find(Common\ Edges)$
4.  $while\ count < g\ do$
5.      $e(u, v) = Random\ select\ edge(Difference\ Edges)$
6.      $Update\ Pnew = Union\ Find(e(u, v))$
7.  $end\ while$
```

Algorithm 5: Mutation Process

Input: Particle P
Output: New Particle

```
1.  $Select\ a\ random\ edge\ e_1 \in P$
2.  $Rest\ Edges = P - e_1$
3.  $A\ new\ particle\ p_{new} = Union\ Find(Rest\ Edges)$
4.  $while\ Pnew\ not\ complete\ tree\ do$
5.      $Select\ two\ random\ points\ p_1, p_2 \in P$
6.      $if\ Find\ set(p_1)\ != Find\ set(p_2)\ then$
7.          $Union(p_1, p_2)$
8.          $break$
9.      $end\ if$
10. $end\ while$
```

Theorem 2: *If g is the number of generations and n is the number of pins then the RRST can be obtained in $O(gn)$ time.*

Proof: The first step of Algorithm 3 is to generate the initial population. If s is the size of population, the obstacle avoidance rectilinear Steiner tree is constructed for s times. So the initial population is created in $O(s\ n \log n)$ time.
The binary tournament and roulette wheel selection operation select one child by comparing two solutions. Both runs the number of the of times the size of population which is $O(s)$.
The crossover and mutation operation selects parents randomly in $\theta(1)$ times but performs the union-find operation in $O(n)$ times as per the Theorem 1. The update operation after a new solution takes $\theta(1)$ times.
The complexity depends on the number of generations and all genetic operators like selection, crossover, mutation, etc. If $g > s$, then the total time complexity becomes $O(s\ n \log n) +$

$O(g\,s) + O(g\,n) \approx O(g\,n).$

4 Experimental Result

The proposed RRST algorithm is implemented in Python language and tested on a PC with Intel(R) Core(TM) i7-6700 CPU with speed 3.40GHz and 8 GB main memory. The current study uses many graph benchmarks from the DIMACS Challenge to demonstrate the practicality of the technique [1]. In this section, the existing literature [5, 9, 10, 12, 19]wire-length and reduction percentage for the obstacle-avoiding problem are compared to RRST. The execution times are also presented and compared with [5, 9]. This method uses fewer non-pins to connect the outliers and generates fewer Steiner points. The number of non-pins and Steiner points in RRST and Guo et al. (PORA) [5] are compared at the final part of this section.

Table 1: Instances from DIMACS Challenge.

Benchmark	Pin#	Obs#
IND1	10	32
IND2	10	43
IND3	10	59
IND4	25	79
IND5	33	71
RC01	10	10
RC02	30	10
RC03	50	10
RC04	70	10
RC05	100	10
RC06	100	500
RC07	200	500
RC08	200	800
RC09	200	1000
RC10	500	100
RT01	10	500
RT02	50	500
RT03	100	500
RT04	100	1000
RT05	200	2000

The Table 1 contains information on the IND, RC, and RT benchmark cases from the DIMACS Challenge, as well as their obstacles and pins. There are five instances IND1-IND5, ten instances RC01-RC10, and five instances RT01-RT05, each with a different number of pins and obstacles. The IND instances include a minimum of 10 pins and 32 obstacles and a maximum of 33 pins and 71 obstacles. The IND instances can contain as little as 10 pins and as many as 32 obstacles, with a maximum of 33 pins and 71 obstacles. Similarly, RC instances have a minimum of 10 pins and 10 obstacles and a maximum of 500 pins and 100 obstacles. The benchmark RT contains large number of obstacles than pins, with a minimum of 10 pins along with 500 obstacles and a maximum of 200 pins with 2000 obstacles.

Table 2: Wire-length Comparison and Reduction Percentage

Benchmark	**Wire-length**						**Reduction %**				
	[5]	[12]	[29]	[9]	[10]	RRST	[5]	[12]	[29]	[9]	[10]
IND1	622	-	639	618	609	145	76.69	-	77.31	76.54	76.19
IND2	9500	-	10000	9800	9691	11598	-22.08	-	-15.98	-18.35	-19.68
IND3	600	-	623	613	613	313	47.83	-	49.76	48.94	48.94
IND4	1109	-	1126	1146	1118	987	11.00	-	12.34	13.87	11.72
IND5	1345	-	1379	1412	1365	1154	14.20	-	16.32	18.27	15.46
RC01	26334	30410	27540	27630	27015	18228	30.78	40.06	33.81	34.03	32.53
RC02	42462	45640	41930	43290	43882	36551	13.92	19.91	12.83	15.57	16.71
RC03	54722	58570	54180	56940	54737	47109	13.91	19.57	13.05	17.27	13.94
RC04	60925	63340	59050	61990	60800	46982	22.89	25.83	20.44	24.21	22.73
RC05	75146	83150	75630	75685	75685	49626	33.96	40.32	34.38	34.43	34.43
RC06	84030	149725	86381	84662	85808	166394	-98.02	-11.13	-92.63	-96.54	-93.91
RC07	113056	181470	117093	113598	113672	130962	-15.84	27.83	-11.84	-15.29	-15.21
RC08	118277	202741	122306	119177	122057	122306	-3.41	39.67	0.00	-2.63	-0.20
RC09	117722	214850	119308	117074	117993	117074	0.55	45.51	1.87	0.00	0.78
RC10	167781	198010	167978	167219	169443	135408	19.29	31.62	19.39	19.02	20.09

Average	9.71	27.92	11.41	11.29	10.97

The Table 2 compares wire-length calculated in earlier reported works [5], [12], [19], [9] and [10] with the proposed method RRST. The 1st column of the table represents benchmark instances, 2nd, 3rd, 4th, 5th, 6th and 7th column shows the Manhattan distance as the wire-length of all other model including RRST. The wire-length reduction percentages with other papers are provided in the 8th, 9th, 10th, 11th and 12th columns respectively. The reduction percentage signifies the difference between the proposed approach and the others. For the benchmarks IND1, IND3, IND4, IND5, RC01, RC02, RC03, RC04, RC05, RC09, RC10 the proposed method RRST has a shorter wire length than previous models. These values are highlighted in bold in the 7th column of Table 2. The last row of the table indicates the average of the reduction percentage. The average wire-length reduction is 14.26% as compared to other models.

In the Table3, the actual execution time is captured using the $time()$ library function. The 1st column contains the benchmark instance details, 2nd, 3rd and 4th columns contains the [9], [5] and RRST respectively. In most benchmarks, RRST has a shorter execution time than [5] but a longer execution time than [9].

Table 3: Comparison of Execution Time

Benchmark	FH-OAOS[9]	PORA[5]	RRST
IND1	0.02	0.57	0.2
IND2	0.02	0.66	1.01
IND3	0.02	0.5	0.2
IND4	0.02	1.32	0.8
IND5	0.03	1.54	1.23
RC01	0.01	0.64	0.54
RC02	0.02	0.25	0.4
RC03	0.07	1.12	1.05
RC04	0.13	2.4	2.09
RC05	0.19	3.68	3.5
RC06	0.31	4.55	6.56
RC07	1.26	8.17	9.43
RC08	2.07	9.34	9.36
RC09	2.43	11.2	10.7
RC10	3.8	20.45	19.5

Table 4: Comparison of Non-Pins Included and Steiner Points Generated

Instances		PORA [5]		RRST	
Benchmark	Pin#	Non-Pin#	Steiner Points#	Non-Pin#	Steiner Points#

IND1	10	5	7	0	7
IND2	10	40	40	35	31
IND3	10	7	12	0	9
IND4	25	18	33	8	29
IND5	33	15	31	12	21
RT01	10	375	189	272	121
RT02	50	645	341	530	294
RT03	100	600	445	513	380
RT04	100	1180	565	1093	692
RT05	200	2851	1567	2700	1453
Average		573.6	343	516.3	303.7

Another goal of this strategy is to minimize the number of Steiner points and non-pin nodes as much as possible. The Table 4 shows the number of non-pins included to connect the outlier pins and number of Steiner points generated by the RRST model against [5]. The 1st and 2nd columns show instances details such as benchmarks and number of pins. The 3rd and 4th columns mention the number of non-pins added and Steiner points generated in PORA [5] model. The 5th and 6th columns present the non-pins and Steiner point produced in RRST model. It can be seen that the number of non-pin depends on obstacles. The last row finds the average of all non-pins and Steiner points. On an average the RRST generates fewer non-pins and Steiner points than PORA[5].

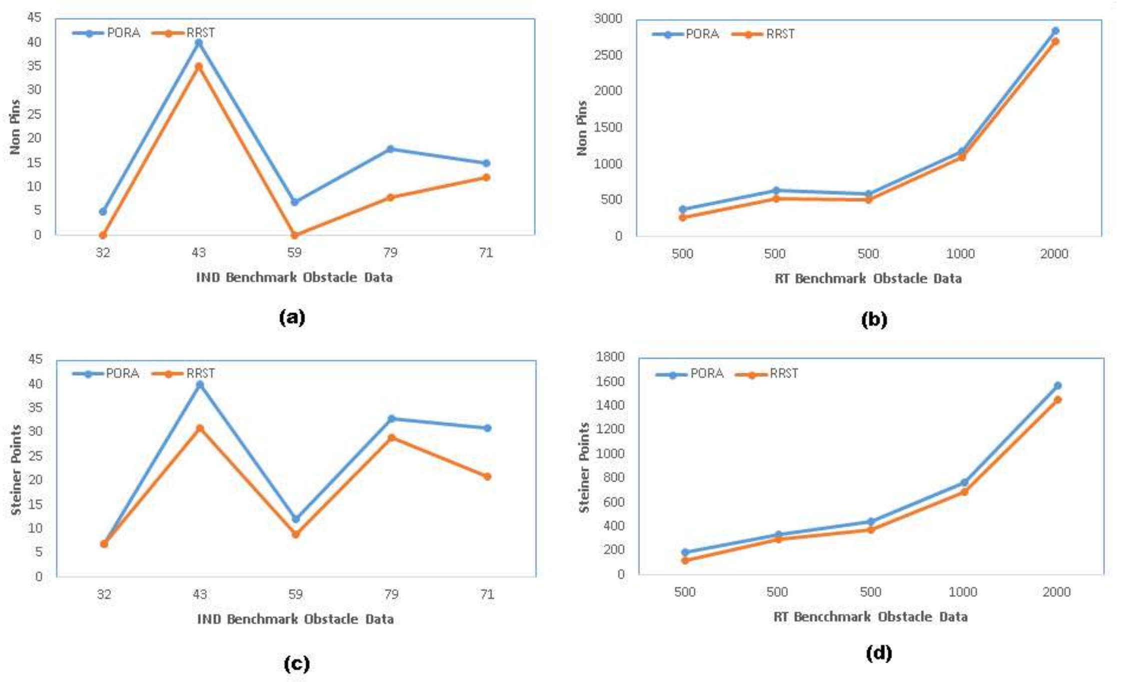

Figure 8: Growth rate of Benchmark data Obstacle, Non Pins and Steiner Points. (a) IND Obstacle Vs Non Pins (b) RT Obstacle Vs Non Pins. (c)IND Obstacle Vs Steiner Points. (d) RT Obstacle Vs Steiner Points.

The Fig. 8 shows the growth rate of non-pins and Steiner Points against the number of obstacles in IND and RT Benchmark Data for PORA [5] and RRST approach. The x-axis represents the obstacles data and y-axis represents the non-pins in Fig. 8 (a) and (b) and Steiner points in Fig. 8 (c) and (d) respectively. In all the figures, the RRST approach line is below the PORA [5] approach. These reduced values indicate the state of the art of the proposed approach RRST which outperforms the earlier works.

5 Conclusion

The current research offers a GA-based optimum obstacle avoidance rectilinear Steiner tree constructing approach. It is a pretty simple and handy approach to build Steiner tree with $O(gn)$ time complexity. The DIMACS Challenge benchmark is extremely difficult, with outlier pins in certain situations. A heuristic approach is developed that adds the fewest possible non-pins to connect the outlier pins. Because there are fewer non-pins, the wire length is reduced by 14.26% compared to earlier reported works with fewer Steiner points. This model will be beneficial for use in the VLSI physical design placement and routing part. Further as the benchmarks contain numerous difficult obstacles, the work can be extended with some real-world application problems in future.

References

1. https://dimacs11.zib.de/downloads.html
2. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms (2009)
3. Fu, Z.H., Hao, J.K.: Swap-vertex based neighborhood for steiner tree problems. Mathematical Programming Computation 9, 297–320 (2017)
4. Ghosal, P., Das, A., Das, S.: Obstacle aware RMST generation using non-manhattan routing for 3D ics. In: Advances in Computing and Information Technology: Proceedings of the Second International Conference on Advances in Computing and Information Technology (ACITY) July 13-15, 2012, Chennai, India-Volume 3. pp. 657–666. Springer (2013)
5. Guo, W., Huang, X.: Pora: A physarum-inspired obstacle-avoiding routing algorithm for integrated circuit design. Applied Mathematical Modelling 78, 268–286 (2020)
6. Guo, Z., Gu, F., Lin, Y.: GPU-accelerated rectilinear Steiner tree generation. In: Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design. pp. 1–9 (2022).
7. Hu, Y., Jing, T., Feng, Z., Hong, X.L., Hu, X.D., Yan, G.Y.: Aco-steiner: Ant colony optimization based rectilinear steiner minimal tree algorithm. Journal of Computer Science and Technology 21(1), 147–152 (2006)
8. Huang, X., Guo, W., Chen, G.: Fast obstacle-avoiding octilinear steiner minimal tree construction algorithm for VLSI design. In: Sixteenth International Symposium on Quality Electronic Design. pp. 46–50. IEEE (2015)
9. Huang, X., Guo, W., Liu, G., Chen, G.: Fh-oaos: A fast four-step heuristic for obstacle-avoiding octilinear steiner tree construction. ACM Transactions on Design Automation of Electronic Systems (TODAES) 21(3), 1–31 (2016)

10. Huang, X., Liu, G., Guo, W., Niu, Y., Chen, G.: Obstacle-avoiding algorithm in x-architecture based on discrete particle swarm optimization for VLSI design. ACM Transactions on Design Automation of Electronic Systems (TODAES) 20(2), 1–28(2015)
11. Iwata, Y., Shigemura, T.: Separator-based pruned dynamic programming for steiner tree. In: Proceedings of the AAAI Conference on Artificial Intelligence.vol. 33, pp. 1520–1527 (2019)
12. Jing, T.T., Feng, Z., Hu, Y., Hong, X.L., Hu, X.D., Yan, G.Y.: λ-oat: λ-geometry obstacle-avoiding tree construction with O(*nlogn*) complexity. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 26(11), 2073–2079(2007)
13. Joshi, B., Thakur, M.K.: Genetic algorithm-and cuckoo search algorithm-based routing optimizations in network-on-chip. Arabian Journal for Science and Engineering pp. 1–10 (2022)
14. Kundu, S., Roy, S., Mukherjee, S.: Sat based rectilinear steiner tree construction. In: 2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT). pp. 623–627. IEEE (2016)
15. Lin, C.W., Chen, S.Y., Li, C.F., Chang, Y.W., Yang, C.L.: Obstacle-avoiding rectilinear steiner tree construction based on spanning graphs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27(4), 643–653 (2008)
16. Lin, K.W., Lin, Y.S., Li, Y.L., Lin, R.B.: A maze routing-based methodology with bounded exploration and path-assessed retracing for constrained multilayer obstacle-avoiding rectilinear steiner tree construction. ACM Transactions on Design Automation of Electronic Systems (TODAES) 23(4), 1–26 (2018)
17. Liu, C.H., Lin, C.X., Chen, I.C., Lee, D., Wang, T.C.: Efficient multilayer obstacle avoiding rectilinear steiner tree construction based on geometric reduction. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 33(12), 1928–1941 (2014)
18. Liu, G., Zhu, W., Xu, S., Zhuang, Z., Chen, Y.C., Chen, G.: Efficient VLSI routing algorithm employing novel discrete PSO and multi-stage transformation. Journal of Ambient Intelligence and Humanized Computing pp. 1–16 (2020)
19. Long, J., Zhou, H., Memik, S.O.: Eboarst: An efficient edge-based obstacle-avoiding rectilinear steiner tree construction algorithm. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27(12), 2169–2182 (2008)
20. Mohapatra, C., Ray, B.B.: A survey on large datasets minimum spanning trees. In: Artificial Intelligence: First International Symposium, ISAI 2022, Haldia, India, February 17-22, 2022, Revised Selected Papers. pp. 26–35. Springer (2023)
21. Muniasamy, R.P., Nasre, R., Narayanaswamy, N.: Accelerating computation of steiner trees on gpus. International Journal of Parallel Programming 50(1), 152–185 (2022)
22. Nath, S., Gupta, S., Biswas, S., Banerjee, R., Sing, J.K., Sarkar, S.K.: Gpso hybrid algorithm for rectilinear steiner tree optimization. In: 2020 IEEE VLSI DEVICE CIRCUIT AND SYSTEM (VLSI DCS). pp. 365–369. IEEE (2020)
23. Nesmachnow, S., Cancela, H., Alba, E.: Evolutionary algorithms applied to reliable communication network design. Engineering Optimization 39(7), 831–855 (2007)
24. Pajor, T., Uchoa, E.,Werneck, R.F.: A robust and scalable algorithm for the steiner problem in graphs. Mathematical Programming Computation 10, 69–118 (2018)

25. Raidl, G.R., Julstrom, B.A.: Edge sets: an effective evolutionary coding of spanning trees. IEEE Transactions on evolutionary computation 7(3), 225–239 (2003)

26. Razafindrazaka, F.H.: Delaunay triangulation algorithm and application to terrain generation. International Institute for Software Technology, United Nations University, Macao (2009)

27. Robins, G., Zelikovsky, A.: Minimum Steiner tree construction. In: Handbook of algorithms for physical design automation, pp. 487–508. Auerbach Publications(2008)

28. Tran, C.: Proposing to improve the heuristic algorithms to solve a steiner-minimal tree problem in large size sparse graphs. Journal on Information Technologies & Communications 2022(1) (2022)

29. Zavoianu, A.C., Saminger-Platz, S., Entner, D., Prante, T., Hellwig, M., Schwarz,M., Fink, K.: Multi-objective optimal design of obstacle-avoiding two-dimensional steiner trees with application to ascent assembly engineering. Journal of Mechanical Design 140(6) (2018)

30. Zhang, H., Ye, D.Y., Guo, W.Z.: A steiner point candidate-based heuristic framework for the steiner tree problem in graphs. Journal of Algorithms & ComputationalTechnology 10(2), 99–114 (2016)