OASALT: On the Construction of Obstacle-Avoiding Steiner shAllow-Light Tree

Wing Ho Lau
CSE Department, CUHK
Hong Kong SAR
whlau22@cse.cuhk.edu.hk

Jinwei Liu
CS Department, HKBU
Hong Kong SAR
jinweiliu@comp.hkbu.edu.hk

Qin Luo
CSE Department, CUHK
Hong Kong SAR
qluo22@cse.cuhk.edu.hk

Evangeline F.Y. Young
CSE Department, CUHK
Hong Kong SAR
fyyoung@cse.cuhk.edu.hk

2025 International Symposium of Electronics Design Automation (ISEDA) | 979-8-3315-3696-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/ISEDA65950.2025.11100533

***Abstract*—In routing tree generation, two important metrics are introduced to evaluate the quality. Wirelength (WL) is directly related to power consumption, routing resource usage and wire delay while pathlength (PL) is indicative of the wire delay. SALT [1] and PD-II [2] are the leading algorithms for the construction of shallow-light routing trees. However, they cannot handle the real designs with blockages. In this paper, we extended the SALT to be Obstacle-Avoiding Steiner shAllow-Light Tree (OASALT) to handle obstacles. First, we improve the time complexity of Lin's [3] Obstacle-Avoiding Spanning Graph (OASG) from $O(n^2logn)$ to $O(n^2)$. In addition, we extended the CL [4] algorithm to generate Obstacle-Avoiding Rectilinear Steiner Minimum Arborescence (OARSMA). The experiment shows that OASALT can achieve a better tradeoff between WL and PL under the cases with obstacles, by combining Obstacle-Avoiding Rectilinear Steiner Minimum Tree (OARSMT) and OARSMA. Compared to performance-driven Obstacle-Avoiding Rectilinear Steiner Tree (PDOARST) [5], OASALT improves the worst delay by 15% and WL by 10% on average.**

***Index Terms*—shallow-light tree, routing, obstacle-avoiding, timing optimization, Steiner tree**

I. Introduction

The generation of routing trees is one of the most essential problems within the field of physical design. The main objective of the routing tree construction is to minimize the total wirelength (WL) and the path length (PL). The WL is intrinsically related to power consumption, routing resource utilization, and wire delay [1], while the PL is indicative of the timing quality. The Rectilinear Steiner Minimum Tree (RSMT) is constructed to minimize WL, with wide application in various tasks including floorplan and placement, global routing, wire length, and timing estimation. A prominent framework for the generation of RSMT is the FLUTE [6] algorithm. The PL can be minimized by employing the shortest path tree methodology. In the context of Rectilinear Steiner trees, the tree is constructed such that all paths from the root are the shortest, achieved through the Rectilinear Steiner Minimum Arborescence (RSMA). Efficient heuristic algorithms, such as those discussed in [4], are employed to construct RSMA.

Optimizing either the WL or the source-to-sink PL in the routing tree generation problem is relatively straightforward. However, enhancing one metric does not necessarily result in the enhancement of the other. Numerous studies turn to the simultaneous optimization of two key objectives: source-to-sink PL (shallowness) and WL (lightness). Prim-Dijkstra [2], [7] and SALT [1] are two main methods in the construction of the Steiner Shallow-light tree. The Prim-Dijkstra algorithm effectively balances the trade-off between constructing the shortest path tree and the minimum spanning tree. The SALT algorithm starts with identifying the break points requiring rerouting in the minimum spanning tree, and utilizes the CL [4] algorithm to route the connections from the source to these break points. Both algorithms exhibit commendable performance in generating shallow-light routing trees.

With advancements in IC technology, contemporary nanometer-scale circuit designs incorporate an increasing number of routing obstacles such as macro cells, IP blocks, modules, and congested regions on 3D-IC [8] or FPGAs [9], [10]. Traditional RSMT and RSMA algorithms are inadequate to address the complexities of modern design routing. Therefore, the implementation of an Obstacle-Avoiding routing tree is essential. Existing works have proposed some solutions to either the lightness-driven or the timing-driven routing tree construction under the obstacles, respectively. The Obstacle-Avoiding Rectilinear Steiner Minimum Tree (OARSMT) is utilized to minimize the wirelength only under the obstacles. Its construction problem has been examined for many years and can be classified into three approaches: 1) maze-routing [11], [12]; 2) escape graph [3]; and 3) correction approach [13], [14]. Li [11] has applied maze routing to address the challenge of OARSMT. His approach involves employing a maze router to determine the shortest paths to successive pins and subsequently utilizing a minimum spanning tree algorithm to select from the identified edges. Chow [12] has improved the maze routing based method and proposed a new parallel approach. The escape graph method [3] constructs a graph that retains information pertinent to circumventing obstacles. In contrast to the maze routing approach, this method significantly reduces both run time and memory consumption by necessitating the traversal of edges only, rather than every individual grid point. Correction approaches like FOARS [13] firstly generate an obstacle-aware Steiner tree (OAST), and then rectilinearize the pin-to-pin connections avoiding obstacles to construct an OARSMT. Regarding the construction of the timing-driven routing tree under obstacles, Lin proposed [5] a performance-driven Obstacle-Avoiding Rectilinear Steiner Tree to improve delay and slack for the Obstacle-Avoiding Rectilinear Tree. However, the trade-off of the WL is huge.

Since limited research concentrates on constructing the shallow light tree for Obstacle-Avoiding Routing, we propose OASALT, a new shallow-light tree construction framework in the Obstacle-Avoiding scenario to improve timing and wirelength simultaneously. Our work improves the result based on 2 major adaptations.

1) We improve the time complexity of Lin's Obstacle-Avoiding Spanning Graph algorithm [3] from $O(n^2logn)$ to $O(n^2)$

2) We extend the CL [4] algorithm as Obstacle-Avoiding CL (OACL) to generate Obstacle-Avoiding Rectilinear Steiner Minimum Arborescence (OARSMA).

Experimental results show that our methods outperform performance-driven Obstacle-Avoiding Rectilinear Steiner Tree (PDOARST) in both wirelength(WL), worst delay (WD) and average delay (AD). Compared to other OARSMT routers, our Obstacle-Avoiding SALT achieves significant WD and AD drops with small WL trade-off.

II. Preliminaries

A. Shallow-light Routing Tree

The routing tree is composed of a set of pins and interconnections between all pins within the Manhattan space. Denote $P = \{p_0, p_1, ..., p_n\}$ as the set of pins of a given net. p_0 serves as the source, while the remaining elements are designated as sinks. Denote $G = \{V, E\}$ as the weighted connected graph, with edge weights corresponding to the Manhattan distances between vertices. A Steiner Tree $T = \{V', E'\}$, $V' \subseteq V, E' \subseteq E$ connects all pins with newly inserted points from V named Steiner Points. In the Obstacle-Avoiding Routing Tree generation problem, vertices V_b and edges E_b do not touch the interior of obstacles. Denote $G_b = \{V_b, E_b\}$, $V_b \subseteq V, E_b \subseteq E$ as the weighted connected graph representing the Obstacle-Avoiding Routing Tree problem. The Obstacle-Avoiding Steiner Tree is constructed under the graph G_b.

There are primarily two objectives in the Shallow-light Routing Tree Construction Problem, namely wirelength (lightness) and pathlength (shallowness) [1], [2]. The wirelength can be quantified by the total weight of the tree. The lightness metrics are often referred to as normalized by the minimum tree, expressed as $\frac{w(T)}{w(RSMT(G))}$ for the Rectilinear Steiner tree, and $\frac{w(T)}{w(RSMT(G_b))}$ for the Obstacle-Avoiding Steiner tree. It is important to note that $w(\cdot)$ indicates the weight of a tree. When optimizing timing, two prevalent metrics are employed: the normalized path length and shallowness. Let $d(\cdot)$ represent the Manhattan distance. The first metric, normalized path length, utilized by PD-II [2], is derived by the total PL normalized by the total shortest-path distance, indicated as $\frac{\sum_{v\in V} d_T(p_0,v)}{\sum_{v\in V} d_G(p_0,v)}$ for the Steiner tree and $\frac{\sum_{v\in V_B} d_T(p_0,v)}{\sum_{v\in V_B} d_{G_B}(p_0,v)}$ for the Obstacle-Avoiding Steiner tree. The second metric, used by SALT [1], is shallowness, which is the maximum ratio of pathlength with its shortest distance , represented as $max\{\frac{d_T(p_0,v)}{d_G(p_0,v)}|v \in P\}$ for the Steiner tree and $max\{\frac{d_T(p_0,v)}{d_{G_B}(p_0,v)}|v \in P\}$ for the Obstacle-Avoiding Steiner tree.

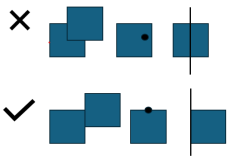


Fig. 1. Obstacles cannot overlap with other obstacles, pins or edges. However, line-touched is allowed.

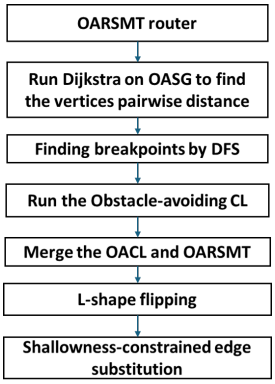


Fig. 2. The workflow of Obstacle-Avoiding SALT

B. Problem Formulation

In the problem of generating Obstacle-Avoiding routing trees, a collection of non-overlapping rectangular blockages and pins is provided. Denote $B = \{b_1, b_2, ..., b_m\}$ as the set of non-overlapping rectangular blockages where $m \in Z^+$. Although these rectangular blockages do not overlap with each other, they may touch at the edges or corners. Let $P = \{p_1, p_2, ..., p_n\}$ represent the set of pins where $n \in Z^+$. All the pins in P are prohibited from being placed inside blockages B, while pressed lines and corners within blockages B are permissible (Shown in Figure 1). The objective is to construct a routing tree minimizing the lightness and shallowness without touching the interior of obstacles [11].

III. Methodology

The overall framework of our work, Obstacle-Avoiding Steiner shAllow-Light Tree (OASALT), is inspired by the SALT [1]. We integrate the Obstacle-Avoiding Rectilinear Steiner Minimum Tree (OARSMT) and Obstacle-Avoiding Rectilinear Steiner Minimum Arborescence (OARSMA) to generate the initial solution. We first generate an OARSMT and select breakpoints that violate shallowness constraints. The breakpoints are connected to the source by OARSMA in order to fix the shallowness violation. It offers a smooth trade-off between the wirelength (WL) and timing-related metrics, including shallowness and normalized path length. By integrating the 2 minimum trees, OARSMT and OARSMA, OASALT efficiently minimizes wirelength with the shallowness constraint.

Figure 2 shows the overall flow of the OASALT. The OASALT starts with the OARSMT. We use FOARS [13] to generate the OARSMT due to its effectiveness in both WL

and runtime. We will run depth-first search (DFS) on the OARSMT to identify breakpoints using the distance calculated on the Obstacle-Avoiding Spanning Graph (OASG) instead of the Manhattan distance since the Manhattan distance may not accurately represent the edge cost between nodes blocked by obstacles. The Obstacle-Avoiding CL (OACL) is used to generate the OARSMA connecting all breakpoints and the source. Finally, L-shape flipping and Shallowness-Constrained Edge Substitution (SCES) are used to further improve WL within the shallowness constraint.

A. Obstacle-Avoiding Spanning Graph

The Obstacle-Avoiding Spanning Graph (OASG) [3] is a conventional approach commonly employed to tackle the Obstacle-Avoiding Rectilinear Steiner Minimum Tree (RSMT) problem. An escape graph is created by connecting the vertices of obstacles and pins, aiding in the determination of paths that avoid these obstacles. By concentrating exclusively on the pins and corners of obstacles, the OASG approach significantly cuts down on computational time and memory usage, offering an advantage over traditional maze-routing methods.

Our graph formulation is similar to [3]. A vertex can be a pin or one of the four corners of the obstacles. Vertex v_1 is considered a neighbor of vertex v_2 if no other vertex belonging to an obstacle resides within or on the edges of the bounding box defined by v_1 and v_2. The OASG is proven to imply a rectilinear shortest path between any two vertices [3].

Initially in the algorithm, we can construct three two-dimensional matrices by converting the routing area into a Grid-graph. The first matrix named *id* serves as a look-up table to determine if the provided coordinate (x, y) corresponds to a vertex or the boundary of obstacles. It helps to get the vertex index immediately given a certain location. Two additional matrices store the distance of the most recent left (and respectively, bottom) object. Algorithm 1 details the process of constructing the distance matrix for the nearest object on the left.

We will execute Algorithm 2 in four distinct quadrants to build the OASG. Algorithm 2 illustrates the process by which we determine the addition of edges for a chosen vertex v located in the top-left area. First, we will set up a bounding box using the vertex v location (x, y) and $(x-\infty, y)$. During the iteration process, we enlarge the bounding box upwards. During the expansion, if the bounding box touches any object, it will move the left boundary to avoid overlapping obstacles. We ensure a bounding box remains free of obstacles to determine if a vertex can be added into the OASG. An edge is added into the OASG when the corresponding vertex lies inside the bounding box during the expansion process. Figure 3 shows an example where we initiate the bounding box. As the bounding box makes contact with the top-right corner of obstacle A and it is the initial vertex, the associated edge is added into the OASG. During expansion, the top-right corner of Obstacle C is encompassed within the bounding box, resulting in the addition of its edge to the OASG. The variable

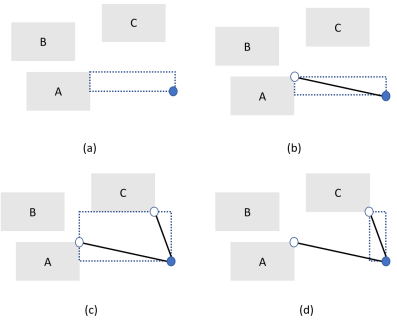


Fig. 3. The example of adding edge of a vertex for top-left corner.

Algorithm 1 Initialize the distance matrix which store the distance for the nearest left object

Input:
dist /*A 2d array storing the distance for the last object on the left*/
id /*A 2d matrix that store the id of vertex given a 2d coordinates*/
Start
0: **for** $y \leftarrow 1$ to $height$ **do**
0: count $\leftarrow 0$
0: **for** $x \leftarrow 1$ to $width$ **do**
0: **if** id[x][y] is a vertex or boundary of obstacles **then**
0: count $\leftarrow 0$
0: **end if**
0: dist[x][y] $\leftarrow$ count
0: count $\leftarrow$ count + 1
0: **end for**
0: **end for**=0

mindist monitors the position of the left edge of the boundary box.

The overall complexity involved in constructing the OASG is $O(n^2)$. Initializing the distance matrix required two nested loop over the x and y axes, resulting in a time complexity of $O(n^2)$. The time complexity for adding edges from a vertex to the OASG in each of the four quadrants is $O(n)$, as it involves 2 single loops, one iteration in the x direction and another one in the y direction. The overall complexity for evaluating all n vertices is $O(n^2)$. The worst-case time complexity, as noted in [3], is $O(n^2 \log n)$ because adding a new Steiner Node can alter the blocking information, making it non-constant in this scenario. Our implementation, as opposed to [3], exclusively operates on a two-dimensional array, enabling parallelization with multi-threading or on a GPU.

B. Obstacle-Avoiding CL

The CL [4] algorithm has been proposed as a solution for the RSMA problem.The basic concept of CL involves repeatedly introducing a new Steiner point to maximize the overlap length for a chosen pair of points. Extending it to include obstacle avoidance is straightforward by employing Grid-graph or OASG. Figure 4 illustrates a scenario of Obstacle-Avoiding CL (OACL) utilizing OASG. Initially, we will execute Dijkstra's algorithm to derive the shortest-path tree on the OASG. At the beginning, all sinks will be added to the active set. Within the loop, back-tracing might be performed on the active node set. Upon discovering a node that has been visited from 2 different sinks, we can determine the maximum overlap of

Algorithm 2 Find the neighbors of vertex v in top-left region

Input:
dist /*A 2d array storing the distance for the last object on the left*/
id /*A 2d matrix that store the id of vertex given a 2d coordinates*/
v(x,y) /*the location of the vertex*/
Start:
0: mindist ← ∞ /* minimum distance */
0: flag ← false /* Check if first vertex is added */
0: **while** (x,y) is not the boundary of obstacle **or** the height of routing region **do**
0: curdist ← dist[x][y] /* current distance */
0: **if** (mindist > curdist **and** flag) **or** (mindist ≥ curdist **and** not flag) **then**
0: **if** id[x-curdist][y] is a vertex **then**
0: Connect vertex[id[x][y-curdist]] and v in OASG
0: $flag \leftarrow true$
0: **end if**
0: **end if**
0: mindist ← min(curdist, mindist)
0: y ← y + 1
0: **end while**=0

Steiner Nodes by examining the related edges. The new Steiner Node will be added to the active set, replacing the 2 initial sinks that will be removed. The process will halt if the active set contains no nodes.

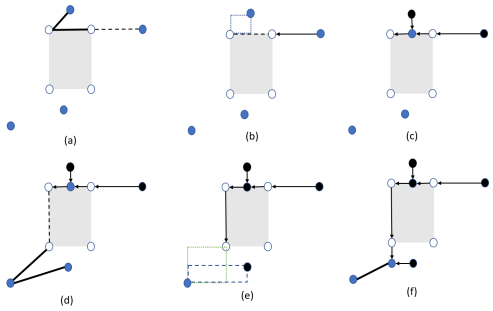


Fig. 4. The blue node is active sink and black node is inactive node. In each iteration, we replace 2 active sinks with new Steiner node that maximize Overlap length. The OARSMA is generated when the set of active sink is empty

OACL might not achieve good performance in OASG when contrasted with the Grid-graph. Figure 5 presents an instance where the performance of OACL on OASG is inferior to that on Grid-graph configurations. In OASG, we overlook situations where two vertices are partly obstructed, preventing us from identifying the optimal Steiner point in each iteration. As depicted in Figure 5b, it is apparent that the maximum overlap Steiner point is positioned at the intersection of the bounding box created by the source with the two specified sinks. However, an obstruction blocks the bounding box, making it hard to locate the necessary Steiner. Thus, we apply a grid graph for OACL, which has more necessary vertices than OASG, to achieve an enhanced OARSMA with reduced wirelength.

C. Post-processing

L-shape flipping and shallowness-constrained edge substitution (SCES) is used to further improve the performance. SCES is proposed by SALT to reduce wirelength without

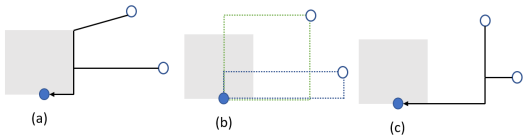


Fig. 5. (a) OARSMA generated using OASG (b) The bounding box formed by source with 2 given sinks respectively (c) OARSMA generated using Grid-graph

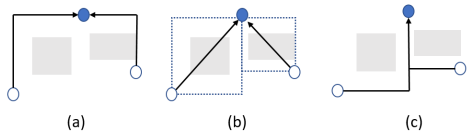


Fig. 6. (a) Before flipping (b) Remove unnecessary Steiner node (c) After flipping

Fig. 7. (a) Before SCES (b) Substitute edge using OACL

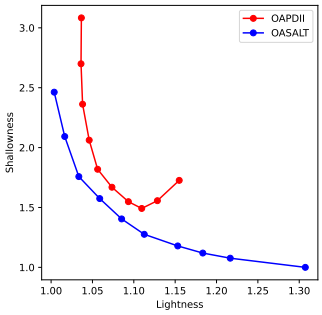
violating the slack constraint. The initial SCES lacks the capability to manage obstacles. We enhance it to address this limitation. Initially, we will eliminate certain Steiner nodes to provide more opportunities for enhancement through L-shape flipping. We run BFS from the leaf node and check the ancestor of the selected node to see if both two L-shape paths can be used and whether the intermediate unnecessary nodes can be deleted safely. A node is unnecessary if it is not a sink and only has one child. The vertex removal should guarantee that the L-shape flipping is safe by checking whether obstacles are on the L-shape path. Figure 6 shows an example of Steiner node removal. The leaf node can be connected to the source using both 2 L-shape paths despite some obstacles being inside the bounding box. Hence, certain Steiner nodes in Figure 6a are deleted before flipping. In SCES, we aim to apply Edge substitution to minimize wirelength while adhering to the shallowness constraint. Nonetheless, a blockage may exist which makes direct substitution of edges impractical. Therefore, we shall perform the OACL process among the target node, candidate node, and the candidate node's parent to identify a path for the candidate node that bypasses obstacles. The goal of employing OACL is to substitute the edge linking the target node to its parent with a shorter edge, ensuring the timing constraint is not breached.

IV. Experiment Results

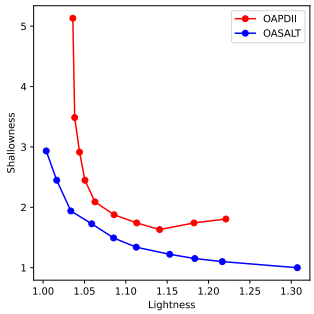
In this section, two sets of experiments were conducted, among which we firstly compared the tradeoff between the lightness and the shallowness, and then validated the superiority of OASALT in the realistic wirelength and delay reduction.

TABLE I
STATISTICS OF OARSMT BENCHMARKS (*pin* DENOTES THE NUMBER OF PINS IN A NET, *obs* DENOTES THE NUMBER OF OBSTACLES)

Case	rc01	rc02	rc03	rc04	rc05	rc06
pin	10	30	50	70	100	100
obs	10	10	10	10	10	500
Case	rc07	rc08	rc09	rc10	rc11	rc12
pin	200	200	200	500	1000	1000
obs	200	800	1000	100	100	10000



(a) *Benchmark I*



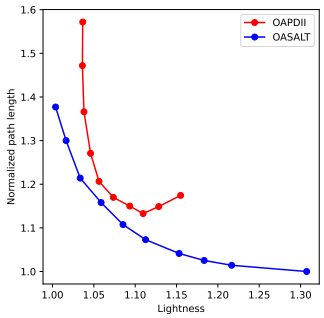
(b) *Benchmark II*

Fig. 8. Shallowness v.s. lightness on different benchmarks

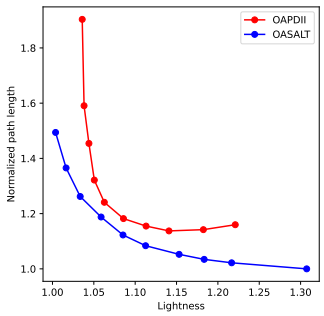
All the algorithms here were implemented in C++ language on a 2.9 GHz Intel Xeon server.

A. Comparison of Lightness and Shallowness

In the first experiment, we would like to evaluate the trade-off curve of OASALT with different ϵ. There are no other works focusing on the Obstacle-Avoiding shAllow-Light Tree. Therefore, we modify PDII to Obstacle-Avoiding PDII (OAPDII) to handle obstacles. We use FOARS to fix the overlapping edge and OASG to compute the distance in OAPDII, with the objective of having a more accurate cost calculation. We randomly generate 2 sets of benchmarks to compare the OASALT with OAPDII. The *Benchmark I* contains 150 nets with 16–32 pins and 32-64 obstacles while the *Benchmark II* contains 150 nets with 64–128 pins and 128-256 obstacles. The Figure 8-9 shows the trade-off between shallowness and lightness, and the trade-off between normalized path length and lightness. α and ϵ are user-defined parameters for the trade-off in OAPDII and OASALT respectively. The parameter



(a) *Benchmark I*



(b) *Benchmark II*

Fig. 9. Normalized path length v.s. lightness on different benchmarks

α varies from 0 to 0.9 with an increase of 0.1, while the ϵ follows a geometric sequence 0.05×1.5^n, where n is an integer ranging from 0 to 9. The lightness is normalized by the wirelength (WL) of the OARSMT, and the normalized path length is calculated as total path length divided by the sum of the shortest source-to-sink distance.

It is obvious from the experiment that the Pareto frontiers of OASALT are much closer to the origin compared to OAPDII. OAPDII uses edge-flipping to iteratively improve the routing tree. However, the edge-flipping cost is hard to construct and may be inaccurate when obstacles exist. OASALT avoids this inaccurate calculation by integrating both OARSMT and OARSMA. Therefore, it can retain the advantage of lower WL in both OARSMT and OARSMA.

B. Comparison of Wirelength and Delay

In the second experiment, we consider wirelength and delay of a net directly. Here we use the benchmark named Obstacle-Avoiding Rectilinear Steiner Minimum Tree (OARSMT), that is widely used in other OARSMT research. It contains 12 cases (rc01-12) with the corresponding statistics in Table I.

We compared the performance of our algorithm to other previous work related to OARSMT and timing-driven Obstacle-Avoiding Rectilinear Steiner tree (OARST). The Maze-routing (MZ) [11] and FOARS [13] are implemented manually in our platform. The result of performance-driven Obstacle-Avoiding Rectilinear Steiner Tree (PDOARST) is directly referenced from the paper [5]. The delay is calculated by the Elmore

TABLE II
THE COMPARISON FOR WIRELENGTH(WL) (um), WORST DELAY (WD) (ps), AVERAGE DELAY (AD) (ps) BETWEEN PDOARST [5] , MAZE-ROUTING [11], FOARS [13], AND OURS OASALT

Case	WL (um)				WD (ps)				AD (ps)			
	[5]	**MZ**	**FOARS**	**OURS**	**[5]**	**MZ**	**FOARS**	**OURS**	**[5]**	**MZ**	**FOARS**	**OURS**
rc01	29140	26040	25980	27780	3384	3906	3810	3191	3008	3149	3093	2773
rc02	42970	42010	42110	45050	4085	4841	3964	3984	3901	3734	3410	3363
rc03	62270	54560	55810	57570	5795	9479	9135	6193	4847	6642	6925	4871
rc04	73870	59590	59350	68290	5908	8073	7880	5228	4819	6154	5978	4466
rc05	87320	74680	74330	78040	6764	13521	10563	6377	6097	10054	7672	5438
rc06	94742	81682	81432	82466	11396	9985	8011	7631	7817	7119	6905	6268
rc07	128875	111733	111129	115990	11016	11834	13399	10379	9725	9223	9992	8004
rc08	137116	116825	117442	122441	11375	15379	15597	8996	10562	12659	13162	7739
rc09	149274	113974	115427	118223	15371	24438	18159	13060	14190	17892	15166	10035
rc10	186030	167440	167500	174090	15815	19265	19798	14252	13758	13801	14691	11534
rc11	253039	235391	233696	240884	32196	51397	62008	19224	22198	38778	44313	16556
rc12	1297170	763569	754942	774459	166205	433910	340686	128261	N/A	317642	242419	88744
Average ratio	1	0.8478	0.8483	0.8904	1	1.4715	1.3556	0.8614	1	1.2153	1.1927	0.8415

TABLE III
RUNTIME(S) BETWEEN PDOARST [5], MAZE-ROUTING [11], FOARS [13], AND OURS OASALT

Case	Runtime(s)			
	[5]	**MZ**	**FOARS**	**OURS**
rc01	0.01	0.03	0.047	0.05
rc02	0.02	0.03	0.05	0.054
rc03	0.03	0.04	0.05	0.062
rc04	0.04	0.05	0.048	0.069
rc05	0.05	0.06	0.05	0.097
rc06	0.45	0.341	0.085	0.46
rc07	0.68	0.422	0.091	0.862
rc08	1.07	0.822	0.119	1.529
rc09	1.93	1.118	0.15	2.112
rc10	0.54	0.135	0.078	2.289
rc11	1.17	0.56	0.109	8.019
rc12	135.31	122.672	6.055	157.844

delay model [15] with the settings as follows: 440 ohms for driver resistance, 0.076 ohms/um for unit wire resistance, 0.118 Ff/um for unit wire capacitance, and 1 Ff for the loading capacitance of the sinks. ϵ is a user-defined parameter for the trade-off in SALT which is set as 1.

We compared both wirelength (WL), worst delay (WD) and average delay (AD). The table II shows the experiment results between our OASALT and other OARST router. Since PDOARST optimizes both timing and WL, we use PDOARST as the baseline. Compared to PDOARST, our OASALT is achieving 10 $\%$ less for WL and 14 $\%$ less for both WD and AD. It demonstrates excellent performance in optimizing delay and WL. Compared to OARSMT router, i.e. MZ and FOARS, the average 30% reduction in worst delay and average delay with the simply 6% increase in WL. The experiment results indicate the higher efficiency and the better tradeoff between total WL and delay reduction achieved by OASALT.

V. CONCLUSION

In this work, we propose Shallow-light Tree for the Obstacle-Avoiding Routing problem to improve timing and wirelength simultaneously. Our approach employs an Obstacle-Avoiding Spanning Graph to have a more accurate calculation for the break points. We adopt Obstacle-Avoiding CL to get OARSMA. We have extended the SALT as OASALT to incorporate obstacle-avoidance capabilities. The results demonstrate that OASALT offers a more balanced trade-off between wire length (WL) and path length (PL) metrics. Additionally, the experimental results show that our router OASALT achieves good performance compared to other OARSMT routers [11], [13] and performance-driven routing tree [5] algorithms.

REFERENCES

[1] G. Chen and E. F. Y. Young, "Salt: Provably good routing topology by a novel steiner shallow-light tree algorithm," *IEEE TCAD*, vol. 39, no. 6, pp. 1217–1230, 2020.

[2] C. J. Alpert, W.-K. Chow, K. Han, A. B. Kahng, Z. Li, D. Liu, and S. Venkatesh, "Prim-dijkstra revisited: Achieving superior timing-driven routing trees," in *Proc. of ISPD*. NY, USA: ACM, 2018, p. 10–17.

[3] C.-W. Lin, S.-Y. Chen, C.-F. Li, Y.-W. Chang, and C.-L. Yang, "Obstacle-avoiding rectilinear steiner tree construction based on spanning graphs," *IEEE TCAD*, vol. 27, no. 4, pp. 643–653, 2008.

[4] J. Córdova and Y.-H. Lee, "A heuristic algorithm for the rectilinear steiner arborescence problem," *Tech. Rep.*, 1994.

[5] Y.-H. Lin, S.-H. Chang, and Y.-L. Li, "Critical-trunk-based obstacle-avoiding rectilinear steiner tree routings and buffer insertion for delay and slack optimization," *IEEE TCAD*, vol. 30, no. 9, pp. 1335–1348, 2011.

[6] C. Chu, "Flute: fast lookup table based wirelength estimation technique," in *ICCAD*, 2004, pp. 696–701.

[7] C. Alpert, T. Hu, J. Huang, A. Kahng, and D. Karger, "Prim-dijkstra tradeoffs for improved performance-driven routing tree design," *IEEE TCAD*, vol. 14, no. 7, pp. 890–896, 1995.

[8] J. Knechtel, E. F. Y. Young, and J. Lienig, "Planning massive interconnects in 3d chips," *IEEE TCAD*, vol. 34, no. 11, pp. 1808–1821, 2015.

[9] W.-K. Mak and E. F. Y. Young, "Temporal logic replication for dynamically reconfigurable fpga partitioning," in *Proc. of the 2002 ISPD*. NY, USA: ACM, 2002, p. 190–195.

[10] M. Hiibner, C. Schuck, M. Kiihnle, and J. Becker, "New 2-dimensional partial dynamic reconfiguration techniques for real-time adaptive microelectronic circuits," in *ISVLSI*, 2006.

[11] L. Li and E. F. Y. Young, "Obstacle-avoiding rectilinear steiner tree construction," in *ICCAD*, 2008, pp. 523–528.

[12] W.-K. Chow, L. Li, E. F. Y. Young, and C.-W. Sham, "Obstacle-avoiding rectilinear steiner tree construction in sequential and parallel approach," *Integr. VLSI J.*, vol. 47, no. 1, p. 105–114, Jan. 2014.

[13] G. Ajwani, C. Chu, and W.-K. Mak, "Foars: Flute based obstacle-avoiding rectilinear steiner tree construction," in *Proc. of the 19th ISPD*. NY, USA: ACM, 2010, p. 27–34.

[14] Z. Shen, C. Chu, and Y.-M. Li, "Efficient rectilinear steiner tree construction with rectilinear blockages," in *ICCD*, 2005, pp. 38–44.

[15] W. C. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *J. Appl. Phys.*, vol. 19, no. 1, pp. 55–63, 1948.