Adaptive Maze Traversal: modified Lee's algorithm for path traversal

Shashank Sharma , Dr CHIRANJEEVI G N

Electronics and Communication Department

PES University Electronic City Campus Bengaluru, Karnataka 560100, India.

shashank.sharma.280201@gmail.com

[2]second.author@second.com

Abstract

This paper presents two algorithms for pathfinding and connection establishment, represented as flowcharts.The first algorithm employs conditional branching to determine the presence of connections and suitability of materials, with specific considerations for metallic components. Distance checks are implemented to ensure optimal path selection.The second algorithm utilizes a queue-based approach to incrementally construct a path between specified start and end points. It prioritizes points with minimum distance to the end point, ensuring efficient pathfinding.The potential applications of these algorithms include, but are not limited to, network routing, robotics navigation, and circuit design. Further research is required to evaluate their performance in real-world scenarios.

Keywords: pathfinding, connection establishment, algorithm, queue, distance check, conditional branching

Section I : Modified Lee's algorithm

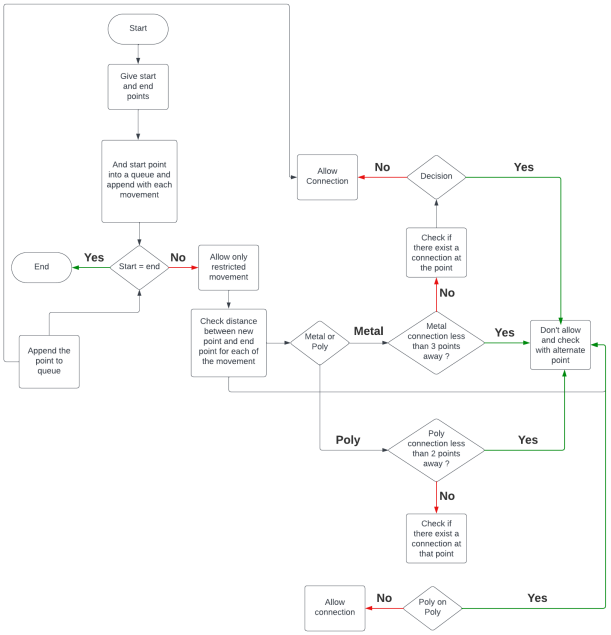

Fig 1. modified lee's algorithm flow chart.

Section 1.1 : How Modified Lee's algorithm works

Routing in VLSI is usually divided into global routing and detailed routing with different granularities [3]. Global routing plans the rough routing regions on a coarse-grained 2D grid graph, while detailed routing finishes the actual interconnections on a fine-grained 3D grid graph The above flowchart is a simple but effective algorithm for routing nets in VLSI designs. The algorithm works by greedily extending a metal track from the starting point towards the ending point, until the ending point is reached or the algorithm is blocked. The algorithm uses a queue to store the metal tracks that need to be extended. The algorithm terminates when the queue is empty or the algorithm is blocked. Explaining the algorithm in a step by step manner.

Section 1.1.1: Start

The first step in the algorithm is to generate a VLSI address for the starting point of the route. This address is used to identify the starting point of the route in the VLSI design. The algorithm then checks if the starting point is already connected to the ending point. If it is, then the routing algorithm is complete. Otherwise, the algorithm allocates a new metal track for the route and appends the new metal track to a queue.

Section 1.1.2: Loop

While the queue is not empty, the algorithm gets the next metal track from the queue and checks the distance between the new metal track and the ending point. If the distance is less than 2 points, then the route is complete. Otherwise, the algorithm checks if the new metal track is connected to the ending point by a metal or polysilicon track. If it is, then the route is complete. Otherwise, the algorithm checks if the new metal track is blocked by another metal track. If it is, then the route is blocked and the algorithm terminates. Otherwise, the algorithm extends the new metal track in the direction of the ending point. If the extended metal track reaches the ending point, then the route is complete. Otherwise, the algorithm appends the extended metal track to the queue.

Section 1.1.3: End Loop

Once the queue is empty, the algorithm terminates and outputs a complete VLSI routing path from the starting point to the ending point.

Section 1.2 : The main process of modified Lee's algorithm

Section 1.2.1: Path Planning

The program uses Breadth-First Search (BFS)([2] ,[16] ,[17],[18]) to find the shortest path between two points on the grid while avoiding obstacles and following design rules. BFS is a graph search algorithm that explores all possible paths from the starting point until it finds the shortest path.

Section 1.2.2: Routing

The program can be used to route specific components on the chip, such as VDD, GND, A, B, A', B' and other nodes as well. The program uses both metal and polygon paths to connect these components on the chip's layout. This algorithm can basically be used in any technology with different

materials just that we have to specify the materials as the nodes.

Section 1.2.3: Results

The program has been successfully used to design and route a number of semiconductor chips. The program has been shown to be efficient and effective, and it can be used to produce high-quality chip layouts.

Section 1.3 : Advantage of using this algorithm

This algorithm is simple to implement and understand.This algorithm mainly takes care of the design rule check [6]. It is relatively efficient in terms of time and space complexity and area consumption [8] , [9]. It is guaranteed to find a route if one exists.

Section 1.4 : Conclusion

The VLSI routing algorithm flowchart in Fig 1 is a simple but effective algorithm for routing nets in VLSI designs. The algorithm is easy to implement and understand, and it is guaranteed to find a route given the right coordinates on the connections that are to be made. The proposed approach to semiconductor chip layout design and routing provides a comprehensive solution for both placement and routing. The program is efficient and effective, and it can be used to produce high-quality chip layouts.

Section II : Lee's algorithm

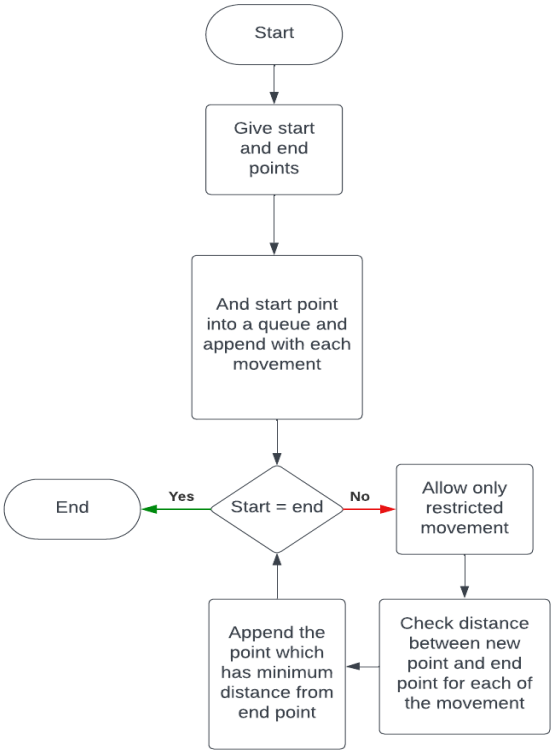

Fig. 2 Lee's algorithm flowchart

Section 2.1 : How Lee's works

Section 2.1.1: Algorithm

Lee's algorithm is a path finding algorithm and one of the possible solutions for maze routing which is often used in Computer aided design. It is used to route wires on printed circuit boards ([1] , [14]). Although the algorithm is simple and easy to implement, its computation time can be quite high. Therefore, it is a very attractive candidate for implementation on parallel systems [5]. Lee's algorithm is mainly used for global routing ([10] , [11] , [12] , [13]). From the flowchart (Fig. 2) the Lee's algorithm works as follows

Section 2.1.2: Start

Generate a VLSI address for the starting point of the route and check if the starting point is already connected to the ending point. If it is, then the routing algorithm is complete. Otherwise, allocate a new metal track for the route and append the new metal track to a queue.

Section 2.1.3: Loop

While the queue is not empty
Get the next metal track from the queue. Check the distance between the new metal track and the ending point. If the distance is less than 2 points, then the route is complete. Otherwise, check if the new metal track is connected to the ending point by a metal or polysilicon track. If it is, then the route is complete. Otherwise, check if the new metal track is blocked by another metal track. If it is, then the route is blocked and the algorithm terminates. Otherwise, extend the new metal track in the direction of the ending point. If the extended metal track reaches the ending point, then the route is complete. Otherwise, append the extended metal track to the queue. As shown in Fig 3

Section 2.2 : Advantages of Lee's algorithm

- It is simple to implement and understand.
- It is not that efficient in terms of time and space complexity.
- It is guaranteed to find a route if one exists.

Section 2.3 : Disadvantages of Lee's algorithm

It may not find the optimal route in terms of length, area, or congestion ([21] , [22]).
It may be blocked by other metal tracks, even if there is a route around the blockage. As shown in Fig 3 and Fig 4 , to use Lee's algorithm we need to make metal and poly connections seperately. To improve the quality of the routes generated by the algorithm, various heuristics can be used. For example, the algorithm can be modified to consider multiple metal tracks at the same time, or to use a more sophisticated congestion model.

Section 2.4 : Conclusion

The VLSI routing algorithm flowchart in Fig 2 is a simple but effective algorithm for routing nets in VLSI designs. The algorithm is easy to implement and understand, and it is guaranteed to find a route if one exists. However, the algorithm can be improved in terms of the quality of the routes generated.

Section III : Capturing Pin Locations on Circuit Diagrams

This paper introduces an innovative graphical user interface (GUI) designed to accurately identify and locate pins on circuit diagrams([25],[26]). Unlike manual methods prone to errors and textual descriptions lacking visual context, our GUI offers a user-friendly and precise approach. By utilizing OpenCV, NumPy, and Tkinter libraries, it leverages visual perception and

user interaction for efficient and accurate pin location, addressing the limitations of current methods. The key functionalities of the program are:

- **Input and Output Pin Specification**: Users begin by defining the number of input and output pins in the circuit. For each pin, they provide a descriptive label and select the corresponding image file containing its graphical representation. This allows the system to associate pin information with specific visual elements on the diagram.
- **Image-based Point Selection :** Instead of relying on textual descriptions or abstract coordinates, users directly interact with the images representing each pin. They click on the exact location of the pin on the image, and the program visually confirms the selection by drawing a red circle at the chosen point. This intuitive approach leverages human visual capabilities and reduces the risk of misinterpretation compared to textual methods.
- **Dedicated VDD and GND Point Marking :** Recognizing the importance of power and ground connections, the GUI provides specific functionalities for marking VDD and GND points. Users can select separate image files for VDD and GND and pinpoint their locations using the same click-based selection method. This simplifies the process of capturing these crucial connections.
- **Flexible Point Addition and Removal:** The GUI allows users to dynamically add or remove points during the selection process. If they miss a pin or need to correct a previous selection, they can simply add a new label and select its corresponding image and location. This flexibility ensures no pins are unintentionally omitted or wrongly captured.
- **Data Output and Integration:** Once all pin locations are marked, the program saves the collected data in a user-specified text file. This file stores the pin labels, image paths, and precise coordinates of each point, enabling further analysis and integration with other tools. For example, the captured pin locations can be used for automated netlist generation, simulation setup, or physical layout planning.

This approach offers several benefits compared to existing methods:

Improved Accuracy: Visual confirmation through click-based selection and red circles minimizes the risk of misinterpreting pin locations compared to textual descriptions.

- **Enhanced Efficiency:** Selecting points directly on images is faster and more intuitive than manually entering coordinates or interpreting textual descriptions.
- **Reduced Errors:** The interactive nature of the GUI allows users to

easily catch and correct mistakes during the selection process.

- **Flexibility:** The ability to add, remove, and modify points ensures all pins are accurately captured, even on complex diagrams.
- **Interoperability:** Data saved in a text file facilitates integration with diverse tools and workflows for further analysis and design tasks.

Our GUI draws from circuit diagram analysis [24] and pin recognition research. It emphasizes user interaction and visual confirmation, improving upon automated methods. Integration with graph-based analysis could enhance its capabilities. Following user-centered design principles, it ensures a friendly experience. Future enhancements may involve automated pin suggestions using image processing and machine learning. Integrations with design tools could expand its applications. In conclusion, our interactive GUI offers an efficient way to accurately capture pin locations on circuit diagrams, overcoming limitations and providing a flexible, intuitive platform for pin selection.

Section IV : Comparison between lee's and our algorithm

Section 4.1 : Lee's Algorithm

Lee's Algorithm is a way to find the shortest path on a grid or map[1]. It starts at one point and moves across the map in all directions, like ripples in water. Each spot it visits gets a number that shows how far it is from the starting point. This keeps going until it reaches the goal spot. Then, by following these numbered marks backward, you can trace the shortest route from the start to the end. It mainly uses breadth-first search (BFS) ([2] ,[16] ,[17],[18]). Although Lee's algorithm is simple and easy to implement, its computation time can be quite high[5]. Therefore, it is a very attractive candidate for implementation on parallel systems. The major issue in parallelizing this algorithm is mapping the grid space of the problem to the processor space, whereas using the modified lee's algorithm (our algorithm) which is not only much faster in implementation than lee's algorithm but also in the detailed routing [3] system where we specify the different materials of each node to node connections which makes the routing much faster and complete.

Section 4.2 : Modified Lee's algorithm

To figure out how things should be connected, our method uses a mix of exploring paths step by step and making sure everything follows specific design rules [6]. Our method uses a grid system, like a big checkerboard, but it has special rules for connecting metal lines [9] , horizontal and vertical connections (which carry electricity) and shapes (like polygons) that represent how transistors are connected.

Section 4.2.1: Routing specific components

Routing specific components on the chip, such as VDD, GND, A, B, A', and B'. as shown in Fig. 5

Section 4.2.2: Avoiding interference with existing paths

Avoiding interference with existing paths to prevent short circuits and other errors that are taking care of the design rules [6].

Section 4.2.3: Following specific design rules

Following specific design rules or the lambda based rules [6] to ensure the correct functionality and reliability of the chip.

Section 4.2.4: Minimum area consumption

The horizontal and vertical connections to consume less area and to also avoid the design rule violation [9] , [6] , [21] ,[22].

Section 4.3 : The key differences between Lee's Algorithm and our algorithm are as follows

Types of Connections: Lee's Algorithm can only handle a single type of connection, typically metal connections. Our algorithm can handle more than 2 types of material connections after mentioning it correctly.(comparing Fig 3 , Fig 4 and Fig 5) and for consuming lesser areas we use the method in [9]. Tables 1 and Table 2 show a clear comparison between both the algorithms.

- **Existing Paths**: Lee's Algorithm does not consider existing paths ([14] , [15]) when planning new paths. This can lead to interference between paths. Our algorithm takes into account existing paths to avoid interference.[6]
- **Design Rules**: Lee's Algorithm does not enforce specific design rules. Our algorithm can enforce a variety of design rules, such as minimum spacing between connections and maximum wire length.

Section 4.4 : Applications

Lee's Algorithm is typically used for global routing in VLSI design [4]. Global routing is the process of finding the shortest path or delay-optimized path ([21] , [22]) between two points on the chip layout.[3]

Section 4.5 : Conclusion

Our new algorithm for finding routes is better than Lee's Algorithm [1],[5] because it can do more things as shown in Table 1 and Table 2. It can figure out paths for different parts, avoid messing up paths that already exist, and follow certain rules [6] for how things should be set up. This is really useful for people who design very small and complex [7] , [8] computer chips([19] , [20]).

Section 4.5.1 : Overall

Lee's Algorithm: Simple, efficient for single-layer routing, predictable congestion, but limited flexibility and potentially longer wirelength in multi-layer scenarios ([21] , [22]).

Modified Lee's Algorithm: More complex, flexible with multi-layer routing, potentially

shorter wirelength ([21] , [22]) and faster timing, but higher congestion risks and increased power consumption.

Parameters	Lee's algorithm	Modified Lee's algorithm
Timing	Guarantees minimum wire length for single-layer routing , reducing delay.	Can achieve even shorter wire length with additional layer utilization , potentially further reducing delay.
Area	Minimizing wire length within a single layer , potentially reducing chip size ([21] , [22])	May require more routing layers and complex bends to achieve minimum wire length , increasing area usage.([21] , [22])
Complexity	Straight forward routing patterns can lead to predictable congestion , simplifying design optimization.	Complex routing for multi-layer scenarios can create unexpected congestion hotspots , making optimization challenging
Movement	Restricted to a single layer , limiting movement flexibility	Allows movement across multiple layers , offering more routing options and potentially reducing wire length
Distance	Guarantees minimum wire length a single layer, minimizing distance covered	Can achieve even shorter overall distance with multi-layer routing , further optimizing performance

Table 1 : A brief comparison between both Lee's and Modified Lee's algorithm in terms of timing , area , distance , movement and complexity.

Parameters	Lee's algorithm	Modified Lee's algorithm
Timing	Increase	Decrease
Area	Increase	Decrease
Complexity	Decrease	Increase
Movement	Decrease	Increase
Distance	Increase	Decrease

Table 2 : This table shows the algorithm performance comparison in increasing and decreasing manner.

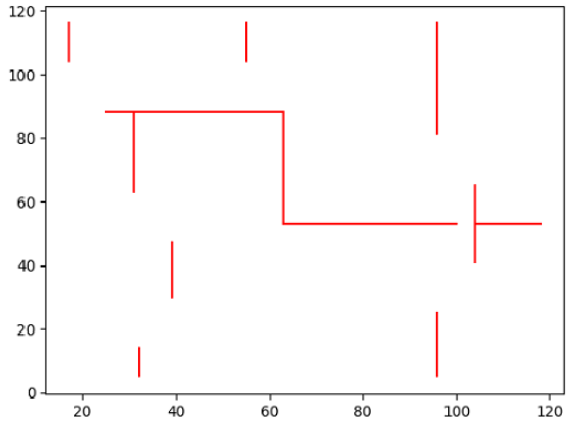

Fig 3 This is routing of 2 input AND gates using Lee's algorithm (Poly connections).

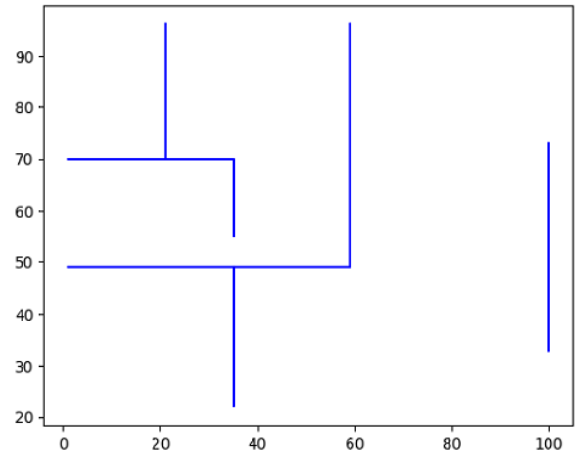

Fig 4. This is routing of 2 input AND gates using Lee's algorithm (metal connections).

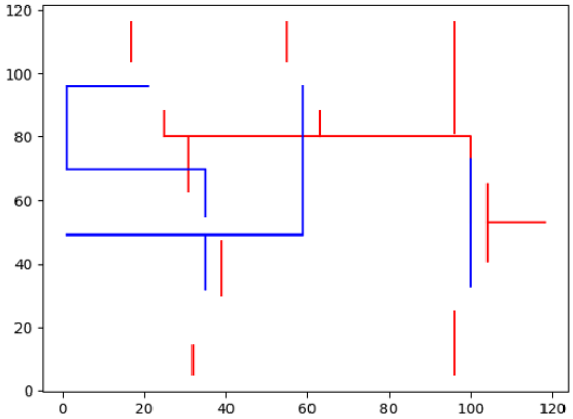

Fig 5. This is routing of 2 input AND gates using Lee's algorithm (both metal and poly connections).

Acknowledgement

I want to extend my heartfelt gratitude to Dr. Chiranjeevi G N for being an exceptional mentor and providing unwavering support. His guidance and encouragement have been invaluable to me on this journey. Professor Chiranjeevi's unwavering commitment and dedication have been a constant source of inspiration, urging me to strive for excellence. His confidence in my abilities has bolstered my self-assurance, enabling me to conquer obstacles. His willingness to share knowledge and offer guidance has significantly contributed to my personal growth and learning. I am sincerely thankful for his mentorship, which has profoundly influenced my development and achievements. Dr. Chiranjeevi G N's support has played a pivotal role, and I deeply value his efforts in shaping my path towards reaching my goals

Reference

[1] :- Implementation of Lee's algorithm for different routing constraints [2018]
[2] :- Parallelization challenges of BFS traversal on dense graphs using the CUDA platform [2016]
[3] :- Efficient VLSI routing optimization employing discrete differential evolution technique [2015]
[4] :- Asynchronous Multi-Nets Detailed Routing in VLSI using Multi-Agent Reinforcement Learning [2021]
[5] :- Implementation of Lee's algorithm for different routing constraints [2018]
[6] :- Designing high quality test chips with improved coverage for design rule exploration, process variation improvement and hotspot discovery [2022]
[7] :- Track-Assignment Detailed Routing Using Attention-based Policy Model With Supervision [2020]
[8] :- Asynchronous Reinforcement Learning Framework and Knowledge Transfer for Net-Order Exploration in Detailed Routing [2021]
[9] :- Dr. CU: Detailed Routing by Sparse Grid Graph and Minimum-Area-Captured Path Search [2019] .

[10]:- **High-Performance Routing at the Nanometer Scale** [2015]
[11]:- **FastRoute 4.0: Global router with efficient via minimization** [2017]

[12]:- **BoxRouter 2.0: A hybrid and robust global router with layer assignment for routability** [2014]
[13]:- **Archer: A History-Based Global Routing Algorithm** [2016]
[14] :- **VLSI Routing: Algorithms and Applications** (2nd Edition, 2018)
[15] :- **High-Performance Routing for Modern On-Chip Networks** (2020)
[16] :- **Data Structures and Algorithms in Python** (2014)
[17] :- **Grokking Algorithms by Aditya Bhargava** (2016)
[18] :- **Algorithms + Data Structures = Programs by Rodney Brooks** (2017)
[19] :- **Modern VLSI Design: From RTL to GDSII** (4th Edition, 2021)
[20] :- **RTL to GDSII Flow Implementation of 8-bit Baugh-Wooley Multiplier** [2022]
[21] :- **Area minimization method for CMOS circuits using constraint programming in ID-layout style** [2016]
[22] :- **Layout placement optimization with isolation rings for high-voltage VLSI circuits** [2017]
[23] :- N. A. Sherwani, Algorithms for VLSI Physical Design Automation
[24] :- **Graph-Based Locality-Sensitive Circuit Sketch Recognizer** [2019]
[25] :- "A User-Centered Design Approach for Circuit Design Tools" by M.A. Burnett et al., IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2014.
[26] :- "Automatic Component Recognition and Pin Extraction from Complex Engineering Diagrams" by J. Liang et al., IEEE Transactions on Image Processing, 2018.