Proceedings of the 5th International Conference on Computing and Data Science
DOI: 10.54254/2755-2721/21/20231148

A comparative analysis of traditional and AI-based routing algorithms in electronic design automation

Zheyi Shen

University Of New South, Wales, Sydney, Australia, 2052

z5456626@ad.unsw.edu.au

Abstract. Rapid advances in artificial intelligence (AI) and machine learning have had a significant impact on various fields, including electronic design automation (EDA). This study aims to compare current EDA routing algorithms with AI-based routing algorithms, highlighting their respective advantages and limitations. Through a comprehensive analysis of existing EDA routing algorithms and artificial intelligence-based technologies, this study explores the applicability of these algorithms in different EDA scenarios, with a focus on their effectiveness and efficiency in routing tasks, and aims to compare and contrast traditional and AI-based routing algorithms in the context of electronic design automation. It also shows that the current AI technology will play an important role in the future development of EDA software. Therefore, the combination of AI technology may be the focus of EDA software development, so as to help elucidate the evolving landscape of EDA and provide insights into the potential future direction, and provide a a comprehensive understanding of their underlying methodologies, advantages, and limitations of traditional and AI-based algorithms in the context of electronic design automation.

Keywords: AI-based routing algorithms, electronic design automation, VLSI routing machine learning

1. Introduction

Electronic Design Automation (EDA), a pivotal area of electronics engineering, has undergone significant evolution over the past several decades. It encompasses a set of software tools that are used to design complex electronic systems such as integrated circuits and printed circuit boards. One of the central tasks in EDA is routing [1], which involves determining optimal paths for electrical connections among different components. Accomplishing this task effectively requires sophisticated algorithms that can balance multiple objectives, such as minimizing path length, reducing congestion, and adhering to design rules.

Traditionally, several established algorithms have been applied to this routing problem, including Maze Routing algorithms, Line Probe algorithms, A* Search Algorithm, Negotiated Congestion, Rip-up and Reroute algorithms, Evolutionary Algorithms, and Simulated Annealing. These algorithms each have their strengths and weaknesses, and their performance can vary significantly depending on the specifics of the task and the computational resources available.

In recent years, a new approach has emerged for tackling routing and other EDA tasks: artificial intelligence (AI). AI, particularly machine learning (ML) and reinforcement learning (RL), offer

DOI: 10.54254/2755-2721/21/20231148

potential benefits such as the ability to learn from data, adaptability to new tasks or changes, and sophisticated handling of multi-objective optimization problems. However, AI-based methods also present their own challenges, including significant data requirements, complexity in implementation, interpretability issues, and high computational demands.

This paper aims to compare and contrast these traditional and AI-based algorithms in the context of EDA. The goal is to provide a comprehensive understanding of their underlying methodologies, advantages, and limitations. Furthermore, the review explores the applicability of these algorithms in different EDA scenarios, with a focus on their effectiveness and efficiency in routing tasks. Ultimately, this analysis may help elucidate the evolving landscape of EDA and provide insights into the potential future direction of this important field.

2. Traditional algorithms

This section provides an overview of the traditional algorithms frequently employed in EDA for tasks such as routing, exploring how they operate and discussing their strengths and limitations.

2.1. Brief explanation of each algorithm

2.1.1. Maze routing algorithms. Maze routing algorithms, such as Lee's algorithm, operate on the principle of treating the design area as a maze to be solved. This algorithm identifies an optimal path from a starting point to an endpoint, traversing through the available routing space. These algorithms are based on graph theory and usually employ a breadth-first or depth-first search to explore possible routes, systematically extending the path until the destination is reached[2]. A typical maze routing algorithm consists of the following steps.

(1) Grid Representation: First, we model the problem as a grid where each cell corresponds to a unit of area on the chip. The starting point (source) and the endpoint (destination) are defined as grid cells. Some cells may be blocked (for example, they might be occupied by other circuit elements).

(2) Initialization: The algorithm begins at the source and labels it with a "0". This label indicates the distance from the source cell.

(3) Wave Propagation (Flooding): Next, the algorithm spreads out from the source, labeling each accessible (unblocked and unlabeled) neighboring cell with an incrementally increased number (representing the distance from the source). This spread is often likened to a wave, hence the term wave propagation.

(4) Backtrace: Once the wave reaches the destination, the algorithm starts there and follows the decreasing numbers back to the source. This backtrace forms the path of least resistance (i.e., the shortest path) from the source to the destination.

(5) Path Selection: The path from the source to the destination, as determined by the backtrace, is then used for the route. If there are multiple possible paths, the algorithm can use other factors to choose between them (such as minimizing the number of turns).

(6) Clearing and Rerouting: If no path can be found because the wave propagation is blocked, the algorithm will need to backtrack, clear some of the path, and try different paths until it finds a path that reaches the destination.

These algorithms are simple, robust, and guaranteed to find a path if one exists. However, they are not the most efficient option for large or complex designs due to their potential to explore a vast part of the entire space, making them slow and computationally expensive.

DOI: 10.54254/2755-2721/21/20231148

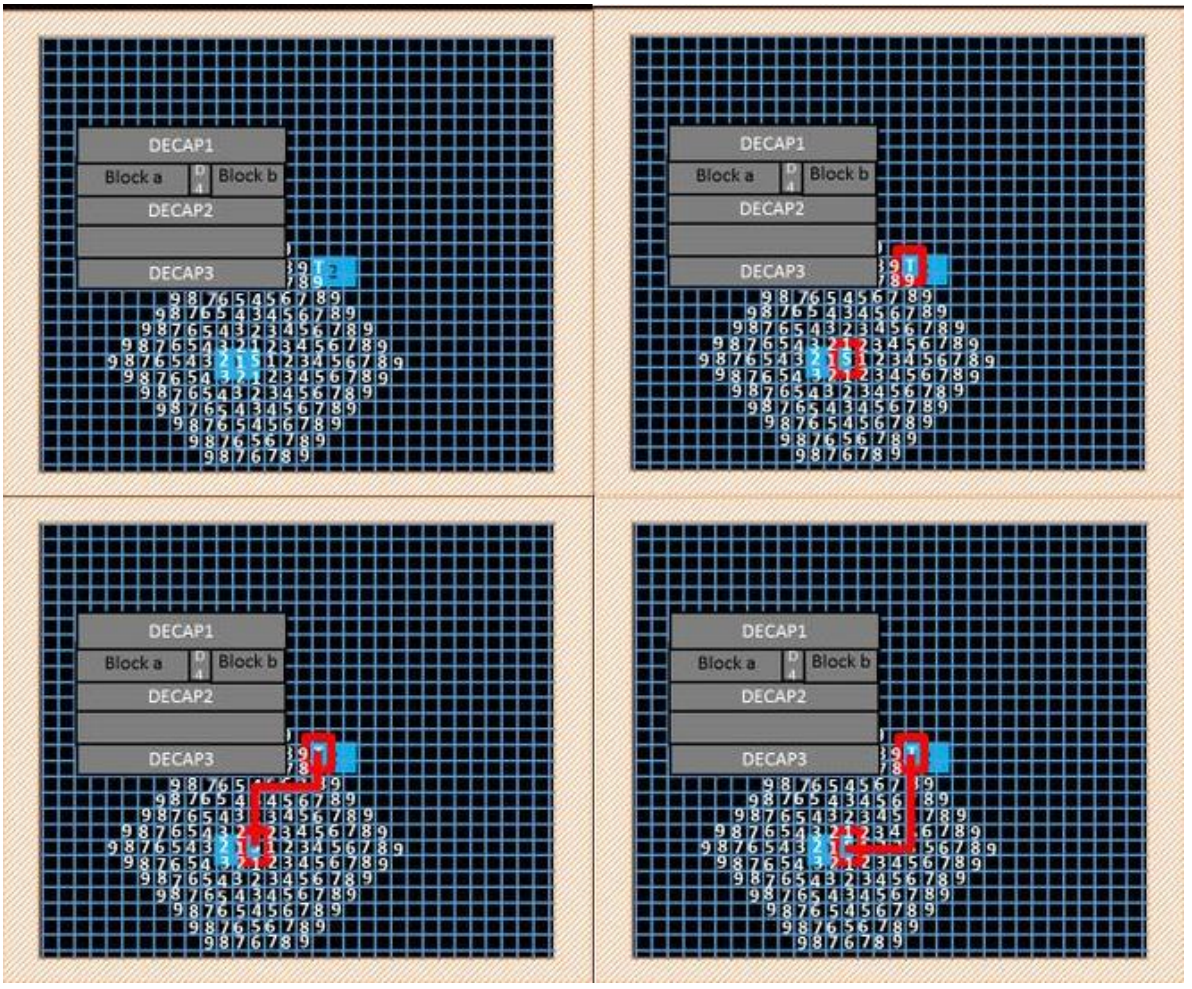

Figure 1. The working process of Maze routing algorithm [3].

2.1.2. Line probe algorithms. Line Probe algorithms is an enhanced variant of Maze routing algorithms, incorporating pattern matching to expedite the search process. This algorithm uses a set of predetermined patterns to guide its search, reducing the space that needs to be explored and speeding up computation. The typical Line Probe routing algorithm consists following steps.

(1) Initial Setup: We start with the starting point S and the destination point D, and we consider all the vertices of the obstacles in the region.

(2) Launching Probes: We launch straight lines (probes) from S towards every vertex of every obstacle. If the probe reaches the vertex without intersecting any obstacle, we connect S and that vertex with a straight line segment.

(3) Adding Intermediate Vertices: For each obstacle that is intersected by a probe, we add the intersection point as an intermediate vertex, and we launch new probes from that point towards all the vertices of all obstacles.

(4) Constructing the Graph: We continue launching probes and adding vertices until all reachable vertices have been connected. This results in a visibility graph, in which each vertex is connected to every other vertex that it can "see" (i.e., reach with a straight line without intersecting any obstacle).

(5) Finding the Shortest Path: Once the visibility graph is complete, we can use a shortest path algorithm such as Dijkstra's Algorithm or A* Algorithm to find the shortest path from S to D.

DOI: 10.54254/2755-2721/21/20231148

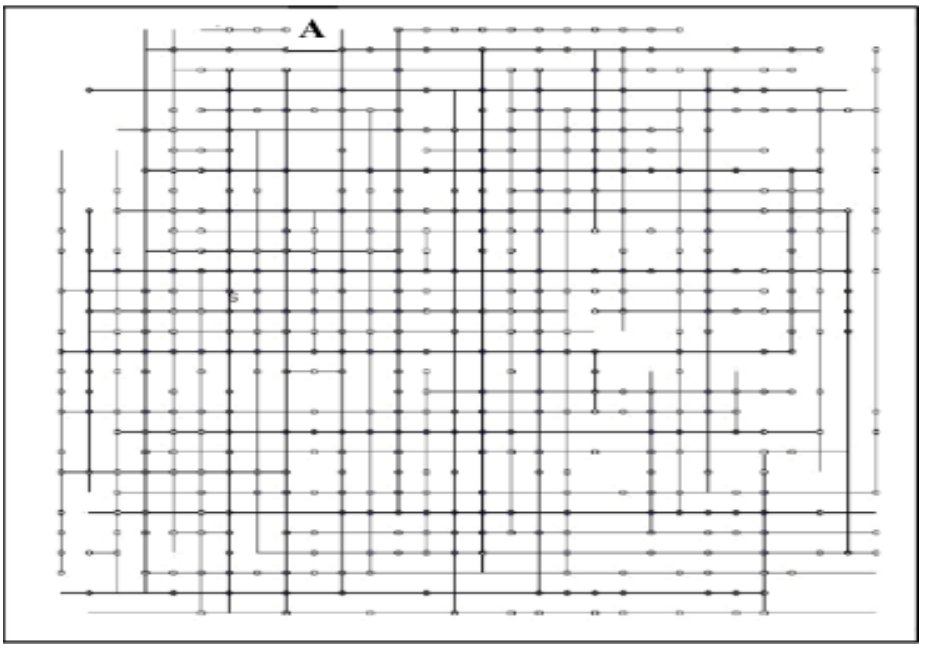

Figure 2. Example of Line Probe Algorithm result [4].

Line Probe algorithms are faster than their Maze routing counterparts due to the reduced search space. However, their performance can heavily rely on the quality of the patterns used, and the implementation can be more complex, due to the need to launch probes from every new vertex towards every obstacle vertex, especially in complex environments with many obstacles.

2.1.3. A search algorithm.* The A* search algorithm is another pathfinding algorithm that is adept at finding the shortest path from a starting point to a destination. A* uses heuristic methods to guide its search, focusing on the paths that appear to be leading towards the destination. It uses a combination of the actual distance travelled and an estimated distance to the goal to prioritize potential paths. In VLSI routing, the algorithm is used to find the shortest possible route for connecting different components without colliding with other components. The outline of the A* search algorithm work process is shown as follows.

(1) Setup: The starting point (source), endpoint (destination), and the chip layout with blocked and unblocked areas (obstacles) are defined. A "cost" grid is created to keep track of the path costs.

(2) Heuristic Function: A heuristic function "h(x)" is defined. This function provides an estimate of the cost of the shortest path from any point "x" to the destination. A common choice is the Euclidean distance or the Manhattan distance between "x" and the destination.

(3) Path Cost Function: A function "g(x)" is defined to represent the cost of the path from the source to any point "x". Initially, this function is set to zero at the source and infinity at all other points.

(4) Priority Queue: A priority queue is set up to store nodes (grid cells) to be explored. The priority of each node is given by "f(x) = g(x) + h(x)", which is the sum of the path cost from the source to "x" and the heuristic estimate from 'x' to the destination. The algorithm always explores the node with the lowest "f" value first.

(5) Algorithm Process: The algorithm starts at the source and explores the neighboring nodes. For each neighboring node, it calculates "g(x)" as the cost from the source to that node (which is the "g" value of the current node plus the cost to move to the neighbor), and 'f(x)' as described above. If this "f" value is lower than the current "f" value for that node, the algorithm updates the "f" and "g" values and sets the current node as the "parent" of that node (i.e., the node from which it's best to move to that node).

(6) Finding the Path: The algorithm continues to explore nodes until it reaches the destination. It then traces back from the destination to the source via each node's "parent", which gives the shortest path.

DOI: 10.54254/2755-2721/21/20231148

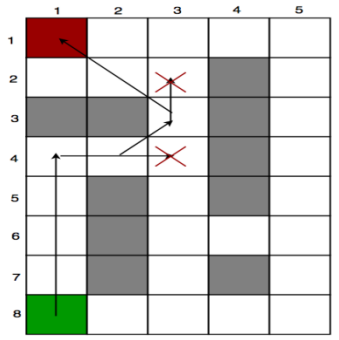

Figure 3. Example of A* search algorithm work process [5].

A* is efficient and effective, capable of finding the shortest path between two points. However, its performance hinges on the chosen heuristic: a well-selected heuristic can lead to quick results, whereas a poorly chosen one can result in slower computation.

2.1.4. Negotiated Congestion and Rip-up and Reroute algorithms. Negotiated Congestion and Rip-up and Reroute algorithms are iterative methods that are especially useful for complex, congested designs. These algorithms start with an initial routing pass, then identify congested areas or regions with violations of design rules. These problematic regions are then "ripped up" and rerouted, and this process may be repeated multiple times for optimization. In the context of VLSI routing, the primary objective is to connect different elements on a chip through a network of wires, while minimizing wire length, reducing the number of layers used, minimizing vias (where a wire jumps from one layer to another), and avoiding wire congestion.

For Negotiated Congestion, this algorithm attempts to prevent congestion from happening in the first place. In each routing pass, it aims to equally distribute the available routing resources among all the nets (a net being a collection of terminals that need to be interconnected). It employs a strategy called "history-based cost", where if a routing resource (like a track or a via) is heavily used in the current pass, it becomes more expensive to use in the next pass. Hence, routers are discouraged from using heavily congested resources, which allows congestion to be shared or "negotiated" between different routes.

Rip-up and Reroute method, on the other hand, allows the initial routing to take place with no or minimal consideration of congestion. After the initial routing is done, if a wire or a set of wires are found to be congested (i.e., they exceed the available routing resource), these routes are "ripped up" or removed. Then, a rerouting process is undertaken to find an alternative route for these wires that doesn't result in congestion. This process is repeated until a valid routing solution is found or until a predetermined limit of rip-up and reroute cycles is reached.

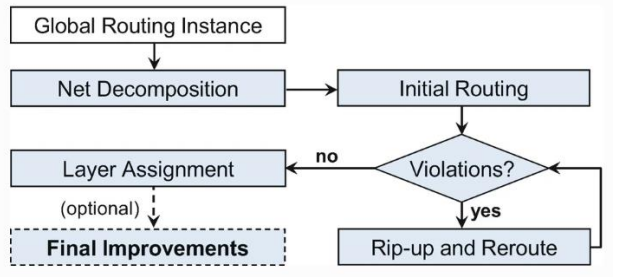

Figure 4. Flowchart of Rip-up and Reroute process [6].

Both of these algorithms have their pros and cons. Negotiated congestion can avoid some cases of congestion, but it can also lead to longer and more complex routes as routers try to avoid congested areas. Rip-up and reroute can potentially find more efficient routes since it doesn't initially avoid congestion, but it can take longer because of the need to repeatedly rip-up and reroute congested wires. In practice, a combination of these strategies, along with other techniques like layer assignment and via minimization, may be used to solve VLSI routing problems.

DOI: 10.54254/2755-2721/21/20231148

2.1.5. Evolutionary algorithms. Evolutionary algorithms, such as genetic algorithms, are optimization algorithms that mimic the principles of biological evolution. Evolutionary algorithms start with a population of random solutions and then iteratively refine these solutions through operations mimicking natural selection, mutation, and crossover. In the context of VLSI routing, an Evolutionary Algorithm may involve the following steps:

(1) Initialization: Create an initial population of routing solutions. Each solution is typically represented as a sequence of steps or a matrix that represents a specific routing configuration.

(2) Evaluation: Assess the quality or fitness of each solution. The fitness function often incorporates factors such as the total wire length, number of vias (transitions between layers), and whether the routing solution violates any design rules, such as wires overlapping or coming too close to each other.

(3) Selection: Choose solutions that will be used to create the next generation. Solutions with higher fitness are more likely to be chosen, but to maintain diversity in the population and avoid premature convergence, some lower fitness solutions may also be selected.

(4) Crossover (Recombination): Generate new solutions by combining parts of two or more selected solutions. This process is analogous to genetic recombination in biological reproduction.

(5) Mutation: Modify some parts of the new solutions at random to introduce further diversity. This can help prevent the algorithm from getting stuck in local optima.

(6) Replacement: Replace some or all of the old population with the new solutions.

(7) Termination: If a stopping condition is met (for example, a solution of sufficient fitness is found, the maximum number of generations is reached, or the population has converged), stop the algorithm. Otherwise, go back to the evaluation step.

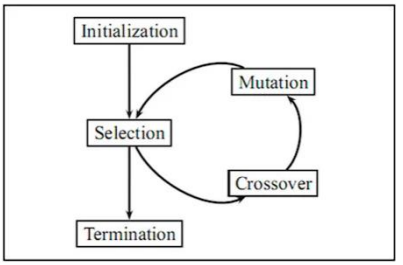

Figure 5. Flowchart of a Evolutionary algorithm[7].

This kind of algorithm can be very effective for VLSI routing because it's capable of exploring a large and complex design space, can deal with multiple conflicting objectives like minimizing wire length while also minimizing the number of vias, and can adapt to find good solutions even as the problem changes, for example, if components are moved around on the chip.

However, the specifics of how to represent solutions, what fitness function to use, how to do selection, crossover, and mutation, etc., can have a big impact on the performance of the algorithm, so careful tuning is often necessary. And even with good tuning, EAs can still take a long time to run on large, complex VLSI routing problems.

2.1.6. Simulated Annealing. Simulated Annealing gets its name from a process in metallurgy known as annealing, where a material is heated and then gradually cooled to reduce its defects and increase its ductility. The "cooling" process in simulated annealing controls the exploration of the search space. Simulated Annealing is a probabilistic technique used for finding an approximation to the global optimum of a given function. It iteratively refines a solution, and at each step, moves to a neighboring solution. The decision to move to a less optimal solution is probabilistic, attempting to escape local minima. Compared with the routing problem of seeking the shortest and best path, simulated annealing algorithm is more suitable for the overall layout planning [8]. The outline of how Simulated Annealing might be applied to VLSI routing is shown as follows.

(1) Initialization: Start with an initial solution. This could be a random routing of all the wires, or it could be the result of a simpler routing algorithm.

DOI: 10.54254/2755-2721/21/20231148

(2) Cost Evaluation: The quality or "cost" of the solution is assessed. This could include factors such as total wire length, number of vias, and violations of design rules.

(3) Generate New Solution: A new solution is generated by making a random change to the current solution. This could involve moving a wire, changing a wire's layer, or rerouting a wire entirely.

(4) Cost Comparison: The cost of the new solution is compared to the cost of the current solution. If the new solution is better (i.e., has a lower cost), it is accepted as the current solution. If the new solution is worse, it may still be accepted with a certain probability. This probability is based on a parameter known as "temperature" and is higher at the start of the process (allowing for more exploration of the solution space) and decreases over time (gradually refining the search).

(5) Iterative Process: Steps 3-4 are repeated for a certain number of iterations or until a termination condition is met, such as a certain level of cost or a maximum number of iterations.

(6) Cooling Schedule: The temperature is gradually reduced according to a cooling schedule. This reduction in temperature reduces the chance of accepting worse solutions, causing the algorithm to make progressively smaller changes to the current solution and converging towards an optimal solution.

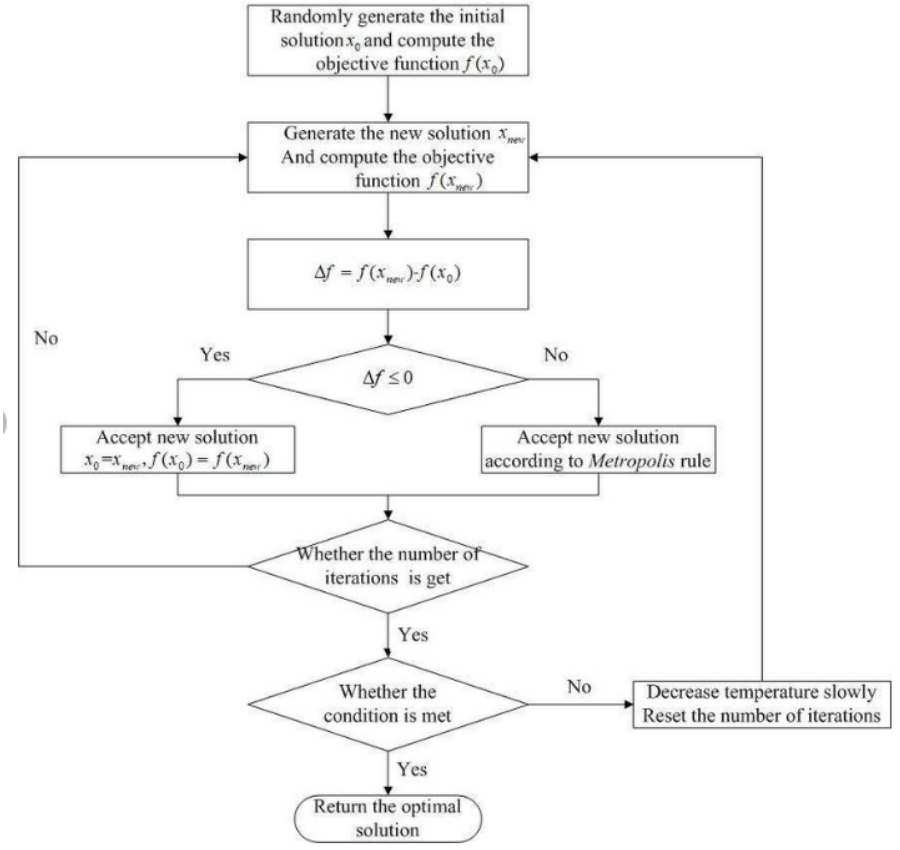

Figure 6. Flowchart of a Simulated Annealing algorithm [9].

The key advantage of Simulated Annealing is its ability to avoid getting trapped in local optima, a common problem with simpler, greedy algorithms. This makes it very effective for complex problems like VLSI routing, where the search space is large and the optimal solution can be hard to find. However, Simulated Annealing is not without its challenges. It can be computationally intensive, particularly for large problems, and the quality of the results can be highly dependent on the chosen parameters, such as the cooling schedule and the method for generating new solutions.

Each of these algorithms offers unique benefits and presents distinct challenges. They have been instrumental in advancing EDA and continue to be valuable tools in the field. The next section will introduce a new class of algorithms—AI-based methods—that bring new possibilities and challenges to EDA.

DOI: 10.54254/2755-2721/21/20231148

3. AI-based algorithms

The advent of artificial intelligence (AI) has opened up new horizons in numerous fields, including EDA. These methods use learning-based techniques that allow them to improve over time and potentially outperform traditional algorithms.

3.1. Introduction to AI and its application in EDA

AI is a branch of computer science aiming to create machines that mimic human intelligence. Machine learning (ML), a subset of AI, involves algorithms that can learn from and make predictions based on data. In EDA, AI methods, specifically ML, have been used for various tasks, including routing, where these methods learn how to make routing decisions based on past data.

3.2. Key concepts

Machine Learning: Machine Learning (ML) is a type of AI that provides systems the ability to learn and improve from experience without being explicitly programmed. ML focuses on the development of computer programs that can access data and use it to learn for themselves. Machine learning models can be trained to predict congestion and other potential issues during the routing phase. This can help guide the routing algorithm to avoid problematic areas and make the overall process more efficient [10].

Deep Learning: A subset of ML, deep learning, mimics the functioning of the human brain in processing data for decision making. It's called deep learning because it uses layered (deep) neural networks to learn from data. Deep learning methods can be used to learn effective routing strategies from large amounts of routing data. For instance, convolutional neural networks (CNNs) can be used to analyze the local routing environment and predict the best direction for routing a wire [11]. Another example can be Graph neural net works(GNN), which is proved to be very efficient on routing and placing [12].

Reinforcement Learning: Reinforcement learning (RL) is a type of machine learning where an agent learns to behave in an environment, by performing certain actions and observing the results/rewards of those actions. Reinforcement learning algorithms can be used to train agents to perform the routing task [13,14]. The agent is given a reward based on how well it routes the design (i.e., minimizing wire length, reducing vias, and avoiding design rule violations) and learns to improve its performance over time based on this reward signal.

3.3. AI-based approaches in EDA

AI-based methods use data-driven approaches to make routing decisions. They can handle complex multi-objective optimization problems by learning from past routing examples and can improve their performance over time. AI-based algorithms involve training a model using a dataset of examples. Once the model is trained, it can make predictions or decisions based on new input data. In the context of EDA, the model could be trained on a range of routing problems and their solutions, and then be able to propose routes for new problems. In addition, several different AI algorithms can be combined to increase the efficiency of the overall system, such as the RL and GNN models mentioned above [15]. AI-based algorithms can adapt to new tasks or changes more easily than traditional algorithms. They have the potential to outperform traditional algorithms as they can learn from data and improve over time.

However, they require a significant amount of data for training, can be complex to implement and maintain, and their decision-making process can be a “black box”, making them hard to interpret. Furthermore, training an AI model can be computationally expensive and time-consuming. In summary, while AI-based algorithms bring new capabilities to EDA, they also introduce new challenges. Understanding these will be key to leveraging their potential benefits and mitigating their limitations in the field of EDA.

4. Comparison of traditional and AI-based algorithms

Routing in EDA is a multifaceted problem that involves a delicate balance between various factors such as minimizing path length, reducing congestion, adhering to design rules, and optimizing other objective

DOI: 10.54254/2755-2721/21/20231148

functions. Different algorithms offer different advantages, and their suitability often depends on the specific nature of the task and the resources available.

4.1. Performance in various routing scenarios

Traditional algorithms like Maze Routing or Line Probe algorithms are deterministic and provide reliable results, making them suitable for simpler, less congested designs. A* Search, Negotiated Congestion, Rip-up and Reroute, Evolutionary Algorithms, and Simulated Annealing are more powerful and versatile, capable of handling more complex routing scenarios, especially those involving multiple objectives or constraints.

In addition, AI-based algorithms have demonstrated a potential to outperform traditional methods in complex routing scenarios, as they can learn and adapt their strategies over time. They can recognize patterns and make decisions based on past experience, allowing them to handle complex, non-linear optimization problems that are common in advanced EDA.

4.2. Efficiency and resource requirements

In terms of computational efficiency and resource requirements, traditional algorithms often have a clear computational complexity and run-time behavior. However, they may suffer from long runtimes or high memory consumption, especially for larger and more complex designs.

AI-based methods, on the other hand, can be resource-intensive during the training phase. They require large amounts of data and computing power to train a model effectively. However, once trained, they can often provide results more quickly for new data, especially in complex routing scenarios.

4.3. Ease of implementation and use

Traditional algorithms tend to be more straightforward to implement and understand. They follow explicit rules and deterministic processes. In contrast, AI-based algorithms require more complex infrastructure for model training and inference. Implementing AI-based methods requires expertise in machine learning and related fields.

4.4. Ability to handle complex and multi-objective optimization problems

Traditional algorithms have well-defined approaches to handle multi-objective optimization problems. However, these methods often need specific adaptations for different types of objectives or constraints. AI-based algorithms, particularly those based on deep learning and reinforcement learning, show a promising ability to handle complex and multi-objective optimization problems more effectively. These algorithms can learn to balance different objectives and handle complex trade-offs based on the data they are trained on.

In summary, while traditional algorithms have been and continue to be the backbone of EDA software, AI-based methods offer exciting new possibilities. Understanding the advantages and challenges of each approach is crucial for their successful application in electronic design automation.

5. Future directions

As we stand at the intersection of traditional and AI-based methodologies in EDA, the path forward promises to be one of integration and evolution.

There is great potential in harnessing the strengths of both traditional and AI-based algorithms in EDA. Hybrid systems, which combine traditional rule-based routing algorithms with AI models, could potentially leverage the consistency and reliability of traditional methods and the adaptability and optimization potential of AI.

Advancements in machine learning techniques also offer promising directions for future research. Transfer learning, for instance, could allow models trained on one type of routing problem to adapt quickly to different types of problems, thereby reducing the time and data required for training. Explainable AI is another important research area that could make the decision-making process of AI models more understandable and transparent.

DOI: 10.54254/2755-2721/21/20231148

Another significant direction for future development is optimizing the efficiency of AI-based methods. The computational cost of training and implementing AI models is currently a major bottleneck. Techniques for reducing this cost, such as model compression, efficient neural network architectures, and hardware accelerators, are areas of active research.

6. Conclusion

This paper has explored the landscape of routing algorithms in EDA, focusing on traditional algorithms and emerging AI-based methods. Each class of algorithms has its strengths and challenges. Traditional algorithms provide robust, deterministic solutions and are often easier to implement and understand, but they may struggle with complex, multi-objective optimization problems. AI-based methods, particularly those based on machine learning and reinforcement learning, offer the potential to handle such problems more effectively. They can learn from data and improve over time, but they require significant resources for training and can be more complex to implement.

As researches move forward, the effective integration of traditional and AI-based methods will likely be a key focus in the field of EDA. This will involve not only technical advancements in algorithm development but also considerations about usability, interpretability, and computational efficiency. As EDA continues to evolve, it will be fascinating to witness the interplay between these methodologies and how they shape the future of electronic design.

References

[1] Ziad Abuowaimer, Dani Maarouf, Timothy Martin "GPlace3.0: Routability-Driven Analytic Placer for UltraScale FPGA Architectures | Request PDF", 12 October 2018, accessed Jun. 09, 2023 https://doi.org/10.1145/3233244.

[2] C. Y. Lee, "An Algorithm for Path Connections and Its Applications," IRE Transactions on Electronic Computers, vol. EC-10, no. 3, pp. 346–365, Sep. 1961, doi: 10.1109/TEC.1961.5219222.

[3] VSD Team,"Maze Routing – Lee's Algorithm – VLSI System Design", accessed Jun. 08, 2023. https://www.vlsisystemdesign.com/maze-routing-lees-algorithm/ ().

[4] W. C.-C. Chu, H.-C. Chao, and S. J.-H. Yang, Intelligent Systems and Applications: Proceedings of the International Computer Symposium (ICS) Held at Taichung, Taiwan, December 12 – 14, 2014. IOS Press, 2015.

[5] "A* Search Algorithm," GeeksforGeeks, Jun. 16, 2016. https://www.geeksforgeeks.org/a-search-algorithm/ (accessed Jun. 09, 2023).

[6] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, "Global Routing," in VLSI Physical Design: From Graph Partitioning to Timing Closure, A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, Eds., Cham: Springer International Publishing, 2022, pp. 131–169. doi: 10.1007/978-3-030-96415-3_5.

[7] D. Soni, "Introduction to Evolutionary Algorithms," Medium, Jun. 23, 2021. https://towardsdatascience.com/introduction-to-evolutionary-algorithms-a8594b484ac (accessed Jun. 09, 2023).

[8] W. Choi and K. Bazargan, "Hierarchical global floorplacement using simulated annealing and network flow area migration," in Automation and Test in Europe Conference and Exhibition 2003 Design, Mar. 2003, pp. 1104–1105. doi: 10.1109/DATE.2003.1253755.

[9] "Figure 1. A flowchart of the simulated-annealing algorithm.," ResearchGate. https://www.researchgate.net/figure/A-flowchart-of-the-simulated-annealing-algorithm_fig1_329917885 (accessed Jun. 09, 2023).

[10] V. Khandelwal, "Machine-Learning Enabled Next-Generation Physical Design – An EDA Perspective," in 2020 ACM/IEEE 2nd Workshop on Machine Learning for CAD (MLCAD), Nov. 2020, pp. 135–135. doi: 10.1145/3380446.3430691.

[11] K.-H. Chang, H.-H. Pan, T.-C. Wang, P.-Y. Chen, and C.-F. C. Shen, "On Predicting Solution Quality of Maze Routing Using Convolutional Neural Network," in 2022 23rd International

Symposium on Quality Electronic Design (ISQED), Apr. 2022, pp. 1–6. doi: 10.1109/ISQED54688.2022.9806227.

[12] D. S. Lopera, L. Servadei, G. N. Kiprit, S. Hazra, R. Wille, and W. Ecker, "A Survey of Graph Neural Networks for Electronic Design Automation," in 2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD), Aug. 2021, pp. 1–6. doi: 10.1109/MLCAD52597.2021.9531070.

[13] H. Handa, "EDA-RL: EDA with Conditional Random Fields for Solving Reinforcement Learning Problems," in Markov Networks in Evolutionary Computation, S. Shakya and R. Santana, Eds., in Adaptation, Learning, and Optimization. Berlin, Heidelberg: Springer, 2012, pp. 227–239. doi: 10.1007/978-3-642-28900-2_14.

[14] U. Gandhi, I. Bustany, W. Swartz, and L. Behjat, "A Reinforcement Learning-Based Framework for Solving Physical Design Routing Problem in the Absence of Large Test Sets," in 2019 ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD), Sep. 2019, pp. 1–6. doi: 10.1109/MLCAD48534.2019.9142109.

[15] N. Wu, Y. Xie, and C. Hao, "AI-assisted Synthesis in Next Generation EDA: Promises, Challenges, and Prospects," in 2022 IEEE 40th International Conference on Computer Design (ICCD), Oct. 2022, pp. 207–214. doi: 10.1109/ICCD56317.2022.00039.