Case Study on Deep Learning based pathfinding model compared to traditional Path Finding algorithm

---

Team Members:

Name	Roll No
Girish S	AM.EN.U4AIE22044
Ashwin Sasi	AM.EN.U4AIE22007
Nandana Krishna MK	AM.EN.U4AIE22034
Harishankar Binu Nair	AM.EN.U4AIE22023

Abstract:

This study investigates the effectiveness and comparative performance of a Deep Learning (DL)-based pathfinding model against traditional pathfinding algorithms by testing it on a generated maze dataset. Comparing traditional pathfinding algorithms and DL models for pathfinding tasks is inherently imbalanced, as DL models must interpret complex image data to learn solutions, whereas traditional algorithms operate on structured and easily represented graph networks or grid systems. To address this imbalance, we standardized the input format by converting maze images into binary matrices, providing both methods with consistent, interpretable data. Using a supervised learning approach, we trained the DL model on these matrices with target movement sequences made up of ‘wasd’ characters, generated by the traditional A* pathfinding algorithm, allowing it to learn effective pathfinding based on labeled data. An unlabeled rectangular image maze dataset, generated by Kruskal’s Algorithm and containing 1000 unique samples, was used for training this model.

This case study explores the DL model’s ability to generalize, its solution speed, and adaptability in comparison to traditional algorithms that rely on structured grid data. We highlight the challenges of fair performance comparisons, as DL models require complex image interpretation, while traditional algorithms operate directly on interpretable grid or graph data. Our results provide insights into the strengths and limitations of each approach, offering perspectives on potential applications and improvements for DL-based pathfinding in scenarios with unstructured or image-based environments.

Problem Statement

The objective of this study is to evaluate the effectiveness and efficiency of a Deep Learning

(DL)-based model for solving mazes, comparing its performance to traditional pathfinding algorithms like A*. While traditional algorithms have shown robust performance in structured grid environments, applying DL-based methods to solve such problems presents new opportunities and unique challenges, particularly when maze representations are complex and image-based. The study aims to develop a DL model that can autonomously navigate mazes represented as images by learning from labeled data generated by traditional algorithms and to assess its performance against established pathfinding methods.

Background

Pathfinding is a fundamental problem in computer science, widely used in robotics, navigation, and game development, where algorithms must identify the shortest or most efficient path between points in a defined space. Traditional pathfinding algorithms, such as A*, Dijkstra's, and Breadth-First Search (BFS), operate effectively on grid or graph structures and are inherently iterative in nature, systematically exploring nodes until the optimal path is found. However, recent advancements in DL present the possibility of creating models that learn to navigate spaces by analyzing complex, less structured data, such as image-based mazes, which represent paths, walls, and openings in pixel-based form. Unlike grid-based systems, image-based environments present greater interpretive challenges, requiring models that can extract, process, and navigate pathways without pre-defined structures. In this study, a dataset of image-based mazes generated using Kruskal's Algorithm serves as a basis for training and evaluating a DL model, leveraging labeled solutions generated by the A* algorithm.

Challenges

1. **Representation Imbalance**: Traditional algorithms and DL models fundamentally differ in how they process data; DL models rely on visual feature extraction from complex image inputs, while traditional algorithms operate on easily interpreted graph or grid structures. This disparity requires careful handling to create fair comparison conditions.
2. **Data Labeling**: Since the DL model requires labeled data, the lack of inherent labels in the maze images necessitates the use of traditional algorithms, like A*, to generate target movement sequences ('wasd' commands) for each maze. This additional step adds complexity and requires reliable labeling to ensure effective DL training.
3. **Generalization**: Traditional pathfinding algorithms are deterministic and provide optimal solutions on structured data, whereas a DL model's ability to generalize to unseen mazes is non-trivial. The challenge is to ensure the DL model can apply learned pathfinding skills across diverse maze configurations with varying wall and path layouts.
4. **Computational Complexity**: While traditional algorithms are optimized for performance and solve problems iteratively, the training of DL models requires significant computational resources. Although training the DL model will take much longer than traditional algorithms like A* or Dijkstra, once trained, **it may predict solutions faster than these iterative methods, presenting a potentially more efficient pathfinding approach in real-time applications.**

Dataset:
Rectangular Maze Dataset via Kruskal's
https://www.kaggle.com/datasets/emadehsan/rectangular-maze-kruskals-spanning-tree-algorithm?resource=download
This study utilizes a dataset of 1000 unique rectangular maze images, generated using Kruskal's Algorithm. Each maze is represented as a 420x420 pixel image, structured in a 21x21 grid. In these images, walls are depicted in black (pixel value 0) and paths in white (pixel value 255). All mazes have a consistent start point at [2, 1] and an end point at [20, 21]. The dataset is publicly available for research purposes and can be accessed at Kaggle

Deliverable Objectives:

- A comparative analysis of traditional Path Finding Algorithms and DL based LSTM networks.
- A Generalized DL model which can solve pathfinding tasks.
- A visual comparison of both DL model and traditional PF algorithms Solution.

Tools and Models Used:

- OpenCV
- TensorFlow
- Matplotlib
- A* Algorithm
- Dijkstra Algorithm