American University of Armenia

College of Science and Engineering

Artificial Intelligence Project

Dynamic Maze

Team: Veronika Khachatryan, Mher Beginyan, Sofi Khachatryan

Fall: 2024

Abstract

This paper investigates various methods for the dynamic maze problem using Artificial Intelligence algorithms. This project mainly concentrates on selecting the pathfinding algorithms that are appropriate for changing environments, where the maze changes after every step that the agent makes. The algorithms included are Greedy, A* Family and D* Lite. These are evaluated considering the methods' dynamic environment adaptability, computational efficiency, and scalability. This research compares these methods and shows their advantages and disadvantages. The research includes the implementation of the best method according to criteria. This work was conducted during an Artificial Intelligence course as a group project in the academic year 2024/2025 at the American University of Armenia.

Keywords: Artificial Intelligence, Dynamic Maze, Pathfinding Algorithms, Adaptability, Scalability, Efficiency, Greedy, A*, D* Lite.

Contents

Introduction **5**

1.1 Origin **5**

1.2 Description **5**

1.2.1 Environment 5

1.2.2 Agent 7

1.3 Structure of the Project Paper **7**

1.4 Goal **8**

Literature Review **8**

2.1 Overview **8**

2.2 Greedy-Best-first Search Algorithm **9**

2.3 Family of A* algorithms **10**

2.3.1 A* with Real-Time Updates 10

2.3.2 D* Lite Algorithm 11

Methods **13**

3.1 Greedy Local Search **13**

3.2 Family of A* Algorithms **13**

Testing and Evaluation **14**

4.1 Overview of Evaluation Metrics **14**

4.2 Experimental Setup **14**

4.3 Performance Analysis of Algorithms **15**

4.3.1 Greedy Local Search 15

4.3.2 A* 16

4.3.3 D* Lite 16

4.3.4 Comparative Analyses 17

4.4 Conclusion **20**

References **21**

List of Figures

Figure 1: Possible Initial State
Figure 2: Possible state after first move.....6
Figure 3: Wall change.....6
Figure 4: Manhattan Distance calculation.....9
Figure 5: Comparison Table.....12
Figure 6: Step count for Greedy algorithm.....15
Figure 7: Step count for A* algorithm.....16
Figure 8: Step count for D* algorithm.....17
Figure 9: Average steps in different types of mazes.....18
Figure 10: Comparative table of step counts.....19
Figure 11: Distribution of expanded nodes.....19

Chapter 1

Introduction

1.1 Origin

Since the inception of civilization, the concept of Maze-like games has been a subject of interest of the greatest minds for centuries, finding its place even in the oldest puzzles known to humanity such as the Labyrinth of Crete in Greek mythology. Historically, mazes appeared as physical structures such as garden or stone mazes, resulting in static and simple environments. However, even then some elements such as moving stones or hidden doors could be included, making the environment more varied and complex. With the development of mathematical ideas and theories, an extra property of optimality was introduced, leading to the birth of algorithms aimed at finding the most efficient paths. Over time, the problem expansion started including dynamic elements as well, making the environment more similar to the real-world scenarios. The invention of computers allowed researchers to find solutions to these challenging problems that require much more complex calculations and analyses. Nowadays, the maze concept with its different variations (restrictions or relaxations) is implemented in many modern games, puzzles, navigation systems, and challenges, great examples of which are the Pac-Man and the Snake games, where enemies or snake's tails represent dynamic obstacles that change their positions over time.

Our Dynamic Maze variation has no specific origin and is defined by our team members. We purposefully chose the dynamic variation to evaluate and compare the performance of different **search and pathfinding algorithms** as its nature allows us to apply the strategy to many real-world simulated environments where changes occur unpredictably, making this project an excellent representation of adaptive problem-solving.

1.2 Description

1.2.1 Environment

Our environment is a collection of cells, stored in a 10x10 grid. The starting position of the agent and goal cell are fixed: (0,0) and (9,9) respectively. The problem will be solved when the agent reaches the goal cell. The grid is a **perfect maze**, indicating that there should be a unique path from one cell to another. To achieve this, we add closed walls (that block our agent's movement in that direction) between 2 adjacent cells (Two cells are adjacent if and only if the Manhattan distance between their positions in a grid is equal to 1). The initial positions of walls are random and visible

to our agent. Moreover, to implement walls' changes we store an invisible indicator in the grid, the movement of which results in a change of walls corresponding to its moving direction and final location. (See figures 1 & 2)

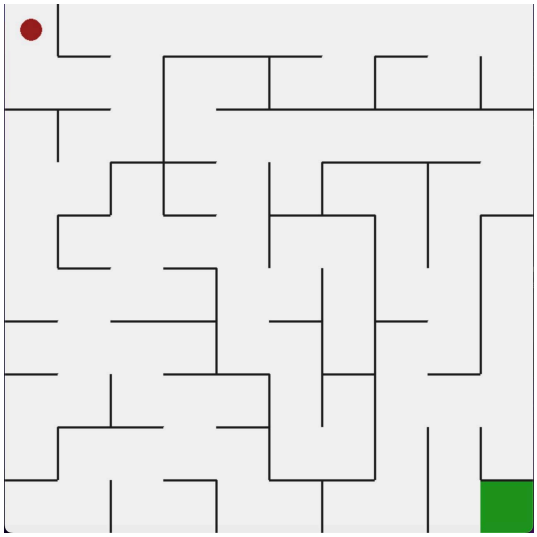

Figure 1: Possible Initial State

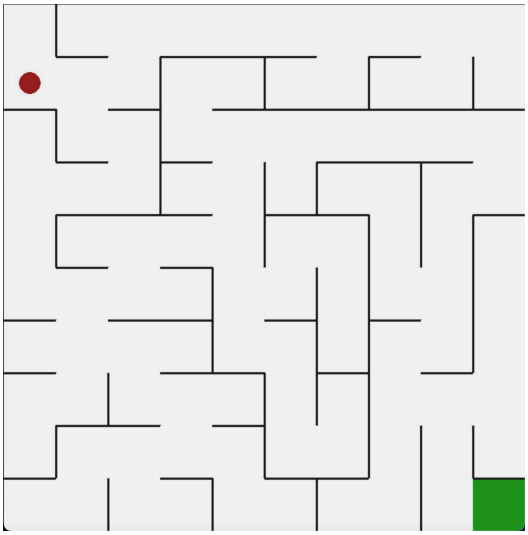

Figure 2: Possible state after first move

To understand the logic of the invisible indicator, particularly how the walls change, let's look at the Harry Potter example above:

Figure 3: Wall change

In this example, the indicator that changes the walls is Voldemort. Voldemort randomly chooses a left, right, or up or down direction. Assume he randomly decided to go left, where there is a wall initially. He will remove the wall, go left, and then add a new one in a location without a wall. Specifically, here, after going left, he will add a wall under his final position, as each node in the

implementation algorithm has a direction, and the direction for that node is “down.” If he chooses to go in a direction where there is no wall initially, for instance, up, nothing will be changed in the maze. This algorithm maintains the perfect “mazeness” of the environment; that is, whenever the walls change, another unique path from the initial state to the goal state occurs.

1.2.2 Agent

In each step, the agent perceives full information about its, goal’s and walls’ positions in the grid. Then based on the specified method it chooses a direction among possible moves and performs that action. After agent’s each move, at most 10 walls are randomly changed, at the same time keeping the property of a perfect maze.

All these properties together make our environment *single-agent, non-deterministic, fully observable, sequential, agent-centric dynamic, and known.*

Moreover, the fixed initial values (maze dimensions, the agent’s and goal positions, changing factor) can be easily changed to adapt the environment to specific real-world scenarios.

1.3 Structure of the Project Paper

This project includes the following sections:

1. Literature Review
2. Method
3. Testing and Evaluation
4. Conclusion
5. Other notes
6. References

The Literature Review includes descriptions and analysis of the potentially efficient methods used in environments similar to our dynamic maze. By reviewing them, we will get insights into their applicability for our specific maze variation and find the best approach that takes into consideration all the features of our dynamic maze.
The method section describes the best approach chosen based on the insights of the previous sections.
The Testing and Evaluation section includes the analysis of the chosen method and evaluates it based on *adaptability, efficiency,* and *scalability*.

The conclusion section includes a summary of key findings connecting the initial purpose of the project to the results achieved during the evaluation phase.
Other notes discuss alternative approaches that are suitable for solving this problem but are not included in the project.
References include a list of materials (articles, papers, videos) used to collect information related to our project.

1.4 Goal

The goal of this project is to test the applicability of different pathfinding algorithms and their various modifications to highlight the advantages and disadvantages of each method for our specific case. We aim to identify and implement the most suitable approach for our dynamic maze ,taking into consideration different evaluation criteria such as **Adaptability** (the ability to adapt the solution to the maze's changes), **Efficiency** (minimizing computational cost and time to find the optimal path), **Scalability** (ability to handle complex or larger mazes). The methods that this project will explore and evaluate will be described in detail in the **Literature Review.**

Chapter 2

Literature Review

2.1 Overview

This section includes a brief summary and analysis of AI algorithms commonly used to solve similar maze problems. This section describes A* with Manhattan distance, D *, and D * Lite algorithms, which have been popular for pathfinding in changing environments. It also summarizes some other methods discussed in the AI course (breadth-first search, depth-first search, Greedy-Best-first Search Algorithm, etc) and explains why these do not apply to the dynamic maze game. It includes a summary of the advantages and disadvantages of each approach in handling our dynamic maze environment, particularly their ability to adapt to the dynamic changes of the maze after each move. The findings of this section are summarized in a tabular format which gives a comparative analysis that is used to guide the selection of the most suitable algorithm for the project.

All algorithms for this project are informed search algorithms, using **Manhattan distance** as a heuristic. The Manhattan distance is the sum of horizontal and vertical moves needed to get from point A to point B through grid-type environments without considering diagonal moves.

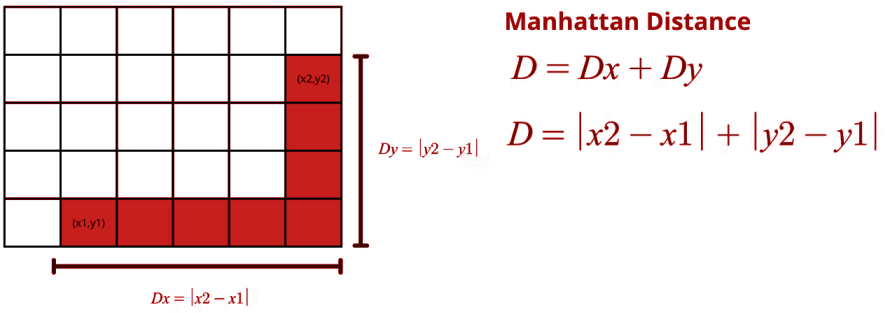

Figure 4: Manhattan Distance calculation

Manhattan distance was chosen as a heuristic as it never overestimates the number of moves required to solve the problem in a grid-structured environment, achieving an admissible and consistent heuristic property.

2.2 Greedy-Best-first Search Algorithm

Greedy algorithms are one of the evolutionary points for AI and have an interesting history rooted in optimization problems, especially in areas like combinatorics and mathematics. The idea of "Greedy" refers to the algorithm's ability to make the best decisions at that moment without worrying whether the current best result will bring the overall optimal result. The first major use of greedy algorithms was in **Huffman coding** in 1952. David A. Huffman introduced this algorithm as part of his work on data compression. The goal was to create efficient codes for data, reducing the size of files without losing information.

Utilizing the Manhattan distance in the greedy algorithm guides the agent to evaluate the distance of its possible position from the goal state and choose the option that minimizes the distance from the goal cell. The agent keeps making local decisions until the goal is reached or no further progress can be made depending on the restrictions(such as limited moves). This approach comes with its advantages and disadvantages. This method is perfect for keeping memory usage low as it doesn't keep track of past actions or possible future actions and only keeps data from the current moment. Additionally, it's time efficient as it quickly calculates the distances and makes quick decisions. However, this method is not quite suitable for dynamic mazes as it doesn't take into consideration the obstacles lying there for future moves, and as the environment changes with each step, the algorithm might get stuck in a position for a long period of time, unable to adapt to new obstacles or

find a more efficient path. Thus, The Greedy algorithm is not adaptive to dynamic mazes since it performs moves that are locally optimal without adaptation to changes provided by new obstacles after each move. It works very efficiently for short-term solutions but can be very inefficient in dynamic conditions since it might result in a waste of moves or loops due to unexpected changes in a maze. Additionally, it's not scalable in the dynamic maze as it requires wider path exploration ,and if we increase the maze's size, it will require more resources.

2.3 Family of A* algorithms

The A* algorithm is a well-known informed search algorithm that combines the concepts of exact path costs g(n) and heuristic estimates h(n), forming the function f(n) = g(n) + h(n) that helps to guide the search and reduce the number of explored nodes, making it an efficient solution for various problems. It was first described in the paper titled *"A Formal Basis for the Heuristic Determination of Minimum Cost Paths"* published in 1968 by Peter Hart, Nils Nilsson, and Bertram Raphael, where navigation through mazes is a part of the main list of problems the algorithm can be applied to (Hart, Nilsson, Bertram, 1968). However, the agent-centric dynamic property of our maze environment requires additional improvements that allow our agent to handle wall changes. The following section will discuss the possible modifications and their applicability to our maze environment.

2.3.1 A* with Real-Time Updates

The most straightforward solution considers re-evaluation of the path after each agent's move. This can be achieved by perceiving the maze's current structure, generating the optimal path using the A* algorithm, moving the agent to the suggested position, and repeating these actions until the goal is reached. This approach ensures that the agent will move along the most optimal path for the current state of the maze at each time moment, which is a great solution to handle changing walls. However, it can be easily noticed that because of these frequent path re-evaluations this method requires a huge amount of calculations. And to fully understand whether the approach is appropriate for our environment or not, let's dive deeper and evaluate the actual time complexity of the algorithm. As the environment is a perfect maze (a unique path for each pair of points), it can be represented as a spanning tree with MN nodes and MN - 1 edges, the branching factor of which can be calculated as b = 2 * number of edges / number of nodes

$$b = \frac{2(MN-1)}{MN} \approx 2$$

Considering the A* algorithm with bad heuristics, at each step the worst time complexity can be represented as 2^d, where d is the Manhattan distance between agent's current and goal states. And now considering the best scenario when no obstacles occur on the found path over time (static maze with the shortest path length of (M-1) + (N-1)),

$$Total = \sum_{d=1}^{(M-1)+(N-1)} 2^d = 2^{(M+N-1)} - 1$$

Based on the result, we conclude that the algorithm performs many extra evaluations even in the simplest cases, which might highly affect the efficiency of the problem for large maze sizes. All these features make the algorithm satisfiable to the criteria of adaptability, at the same time reducing its efficiency and scalability properties.

2.3.2 D* Lite Algorithm

D* Lite is an alteration of the D* algorithm, which combines incremental and heuristic search methods that use heuristics to focus its search and reuse information driven from previous searches to find solutions to a series of search tasks much faster than it is possible by solving each task from scratch. D* Lite is shorter than D* and employs a single tie-breaking measure to compare priorities. It does not need nested and complex if-statements that occupy up to three lines each (Koeing, Likhachev, 2001). The algorithm repeatedly determines the shortest paths between the current node of the agent and the goal node as the walls change while the agent moves toward the goal state. D* Lite searches from the goal state to the start point, and thus, its g-values are estimates of the goal distances.

In our Maze game, D* Lite, unlike A* with Real-Time Updates and other search algorithms, recalculates paths progressively. That means, due to the wall changes if only a fraction of the environment varies, only the relevant path portions are updated. The algorithm ends path computation early if wall alterations do not influence the optimal path to the destination (goal state). Those fewer updates result in considerable time savings, providing the main advantage of the search algorithm. The method ensures that its path is near-optimal or even optimal by combining heuristic estimations with the cost of previously investigated paths. Considering that A* with Real-Time Updates did not track previously found paths (as it recomputes an optimal path after each step), the agent in D* Lite must keep track of previously explored paths, which already presents the disadvantage of the algorithm being more space and memory-consuming. Overall, the algorithm's ability to adjust its search depending on constantly coming, up-to-date environmental feedback about changing walls makes it reasonable for solving our game. The game demands adaptability in a dynamic (continually changing as the agent moves, not while the agent is thinking) environment, which makes the D* Lite algorithm the most suitable one.

Algorithm	Handles Changes	Cost	Memory Usage	Optimality	Applicability
Greedy-Best-first	No	Low	Low	No	Success depends on the factor of randomness (how the maze is changed)
A* with Real-Time Updates	Yes	High	Moderate	Yes	Suitable for finding optimal paths in frequently changing large environments
D* Lite	Yes	Low	Low	Near Optimal	A more efficient version of A*, suitable for dynamic maze environments where changes are frequent and regional

Figure 5: Comparison Table

Chapter 3

Methods

To solve this problem, our team has developed a simulation of the dynamic maze, the parameters of which were described earlier (See Description 1.2). Those parameters are flexible and can be easily changed to simulate various scenarios. However, they were fixed to reduce the randomness factor while analyzing different solving approaches. The methods we implemented to solve the problem are based on the concepts behind previously described algorithms (See Literature Review).

3.1 Greedy Local Search

The first approach is the implementation of Greedy Local Search with Manhattan distance as its evaluation function. The algorithm takes all the directions available for our agent at the current step, calculates the Manhattan distances of those neighbor cells, and chooses the one with minimal value. In the case of a tie, the agent selects a random cell among the best neighbors. Note that the agent is allowed to move to the neighboring cell even if the neighbor's Manhattan distance exceeds the current cell's value. Although it may result in multiple repetitions of the agent's location, this property ensures that the agent always makes a move, slightly modifying the maze after each step. Based on this, the algorithm will eventually solve the problem if there is no time or step count limitation.

3.2 Family of A* Algorithms

The following two algorithms are based on the concept of recalculating the path to the goal and moving along it. Note, as there is a unique path between any 2 points in our perfect maze environment, any complete pathfinding algorithm (BFS, DFS, Greedy Best First…) could be chosen, resulting in the same steps as the solution. Considering this fact, the A* algorithm was preferable as it has the least time complexity of calculating the path.

So, the difference between the last two algorithms is the frequency of recalculations. The first approach is implementing 2.3.1 A* with Real-Time Updates, which evaluates the optimal path after each agent's move. At the same time, the second algorithm is based on the concept of 2.3.2 D* Lite, which moves the agent along the found path until an obstacle occurs on it, only after which it draws a new path.

After understanding how each algorithm works, we need to perform analyses, results, and conclusions to choose the best approach to solving our problem, which will be discussed in the next chapter.

Chapter 4

Testing and Evaluation

4.1 Overview of Evaluation Metrics

For the evaluation and comparison of methods we take into consideration several matrices. These evaluation matrices account for effectiveness, efficiency, and adaptability within the dynamic maze environment. Matrices include:

- **Average number of steps to reach the goal:** This metric evaluates the average number of steps the agent takes to reach the goal. It shows how efficiently an algorithm finds a path and detects which algorithm finds the optimal solution.
- **Computational Cost:** Accounts for a number of nodes expanded during the path search. It is a very important measure of the efficiency of the algorithm in terms of computational resources, as the number of expanded nodes is usually directly proportional to the execution time

4.2 Experimental Setup

As discussed earlier, to test the algorithms, a dynamic maze environment is designed and implemented in Python and for the analyses of the results, R programming language is used. In the dynamic maze, A*, D* Lite, and Greedy Local Search were tested, each executing 1,000 independent runs for different maze sizes and different amounts of wall changes. To ensure equal conditions, the identical random seed was set. For each run the data of step numbers, time taken, and path recalculations are collected in the format of CSV. The collected data is imported into R and analyzed using data manipulation and visualization techniques.

4.3 Performance Analysis of Algorithms

4.3.1 Greedy Local Search

Greedy Local Search with Manhattan heuristic prioritizes closeness to the goal over optimality, which is reflected in our analyses.

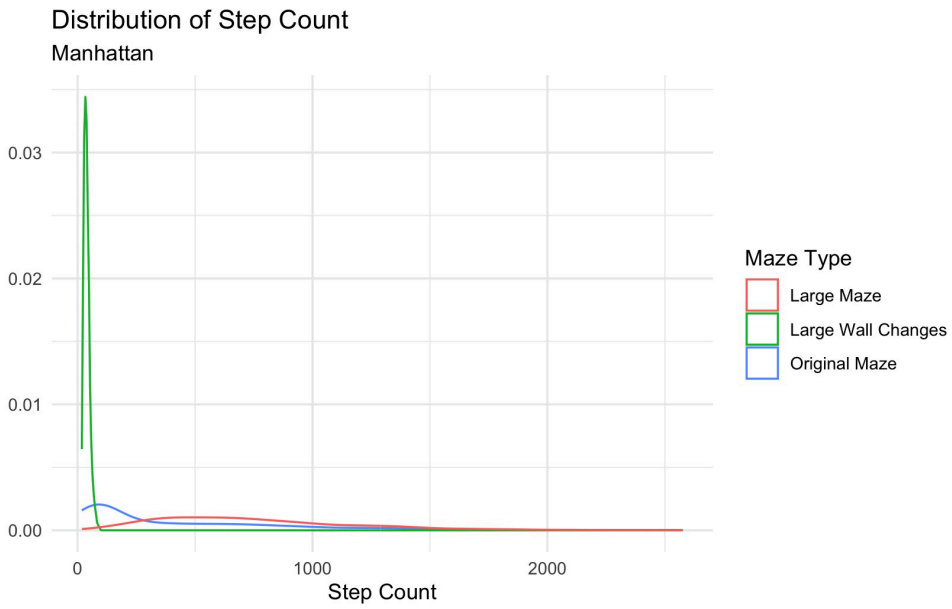

Figure 6: Step count for Greedy algorithm

Figure 6 shows the distributions of the step count of the Manhattan algorithm in different circumstances: in a small original maze, an enlarged maze, and in the original maze where the walls change the maze entirely. For a small 10x10 maze, the average number of steps is about 100 steps. However, with the increase in maze size, the number of steps drastically increases. For example, when the maze dimensions are doubled to 20x20, the average number of steps increases to about 760 steps. This shows the scalability challenges faced by the greedy algorithm. At the same time, in the 10x10 maze, where the walls change more frequently and produce an entirely new maze, we can see that Manhattan performs pretty well, having low step counts with high density (occurrences).

4.3.2 A*

A* search algorithm prioritizes optimality over space. That means that if there is no space limitation, A* will be a good choice among the 3 algorithms, as the agent will compute the optimal path every time it moves and changes the current node. The numbers are reflected in the analysis and the plots below:

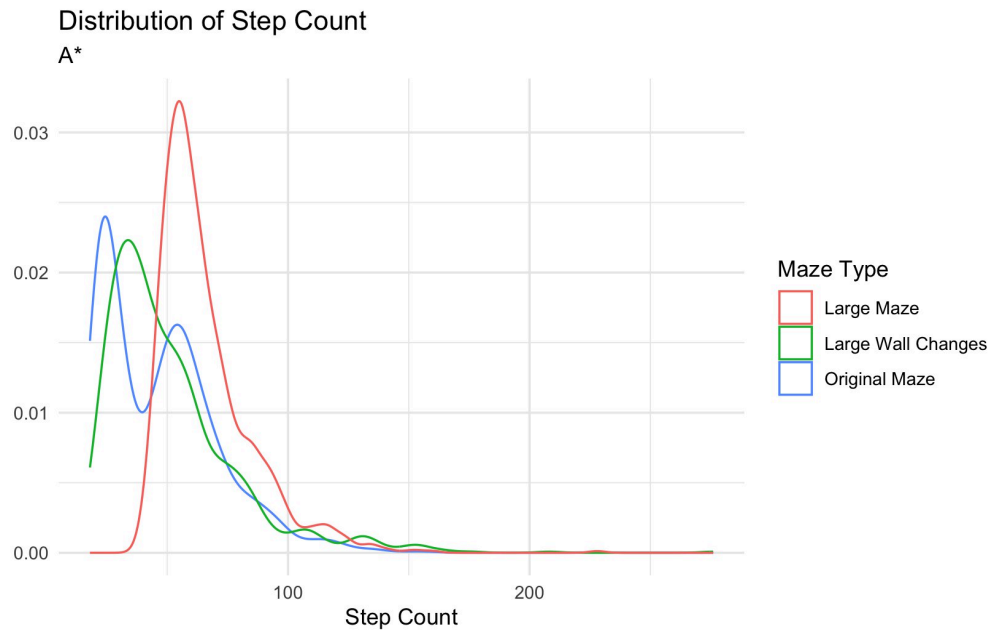

Figure 7: Step count for A* algorithm

Figure 7 shows the distributions of the step count of the A* algorithm in the same circumstances as mentioned above. With the increase of maze size from 10x10 to 20x20, the number of steps increases, and the average number of steps becomes about 66 from 28. However, given that we increase the maze 4 times, the average step number increases by about 2-2.5 times, which is a decent result. This shows the advantages of the A* algorithm. As well as, while comparing the original mazes in both wall-change cases, we can see that while the maze changes entirely, the number of steps does not increase highly. We increase the wall changes 10 times, but the mean number of steps becomes from 28 to 51, again presenting a good result.

4.3.3 D* Lite

D* Lite (Dynamic A*) algorithm is based on the principles of A* with enhanced ability to adapt to dynamic environmental changes. The main difference between A* and D* Lite algorithms is that D* recalculates the path to the goal state **ONLY** when it encounters an obstacle.

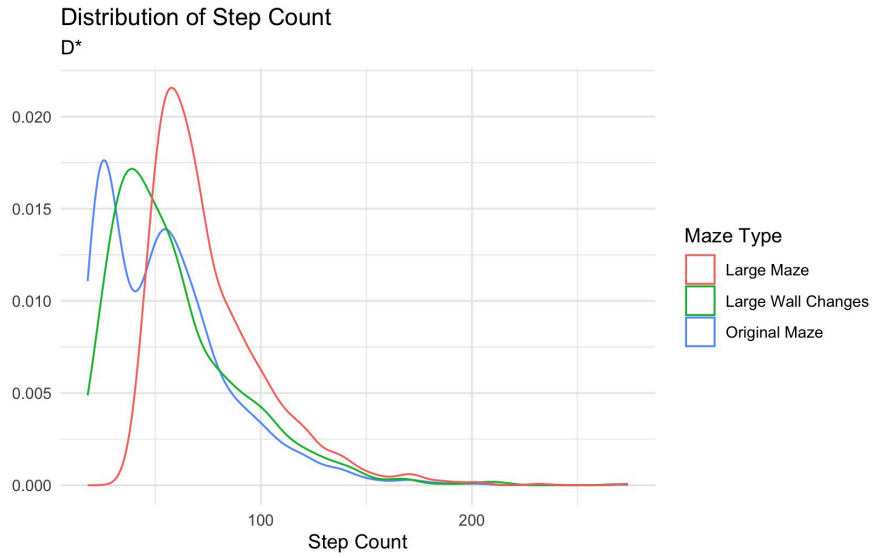

Figure 8: Step count for D* algorithm

As reflected in Figure 8, the average number of steps for a 10x10 grid is approximately 34, which indicates good performance. In a larger, 20x20 maze, the average number of steps reaches 70. This demonstrates the efficient scalability of the D* algorithm to different-sized matrices. Similar to A*, comparing the algorithm's performances in the original maze, where the walls change 10 times, and in the original maze, where the walls change 100 times, the average number of step counts becomes approximately 59. Therefore, here we are having an increase of the mean by less than 2 times.

4.3.4 Comparative Analyses

Further analyzing the algorithms, we have set up different configurations of the maze. This analysis gives insights into how well each algorithm adapts to a dynamic environment.

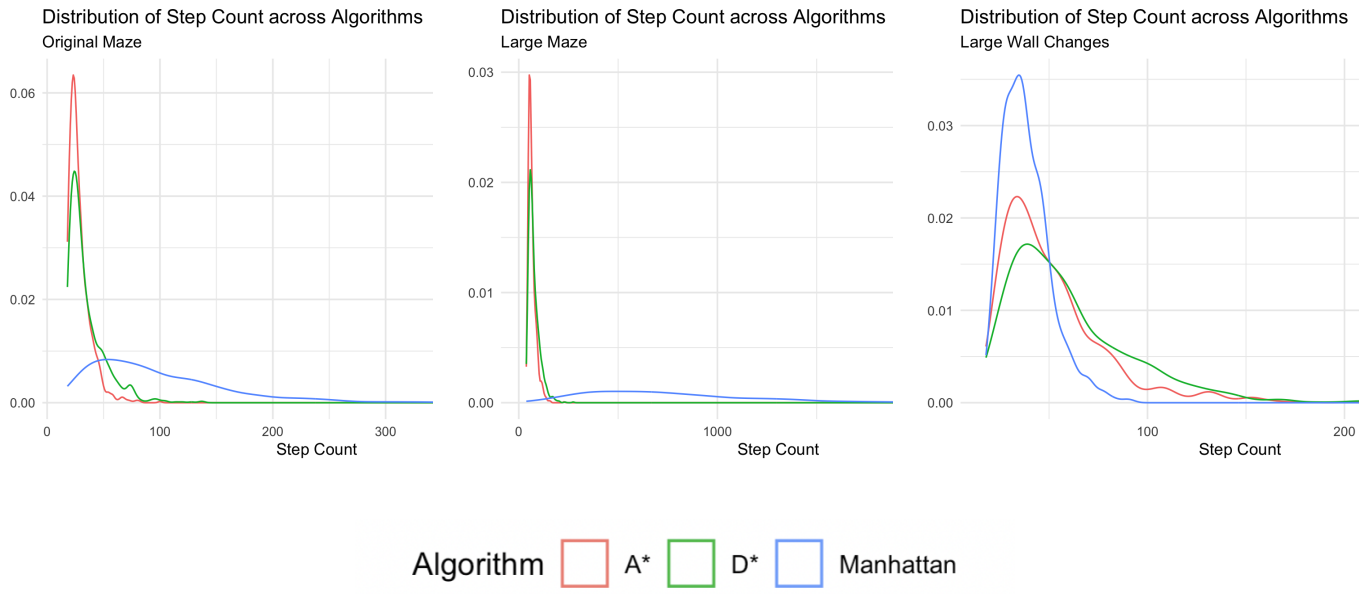

Figure 9: Average steps in different types of mazes

From the findings illustrated in Figure 9, we can conclude A* and D* Lite algorithms perform the best in the original small maze. They show consistent and good performance in different-sized mazes. On the other hand, the Manhattan algorithm performs greatly in a small maze where the walls change more frequently. Indeed, as it focuses directly on the goal cell and tries to move closer at each step, while A* and D* consider the whole path, which might result in getting more distant from the goal. In conclusion, while the Greedy Local Search algorithm excels in highly dynamic mazes due to its fast and goal-oriented nature, A* and D* are performing well, remaining competitive when adaptability and consistency are needed.

The table shown below gives a comprehensive summary of the number of steps for each algorithm under different conditions, including small, large, dynamic, and fully dynamic mazes. It proves the insights gained from the graphical analysis.

Algorithm	MazeSize	WallChanges	AverageStepCount
A*	Large	Small	66.132
A*	Small	Large	51.414
A*	Small	Small	28.690
D*	Large	Small	76.112
D*	Small	Large	59.882
D*	Small	Small	34.750
Manhattan	Large	Small	752.300
Manhattan	Small	Large	38.456
Manhattan	Small	Small	96.606

Figure 10: Comparative table of step counts

For the path-searching algorithms A* and D*, the **number of expanded nodes** is also an important indicator of efficiency and gives insight into the algorithm performance and computational requirements. Thus, we have conducted an analysis to assess this, and the findings are shown below.

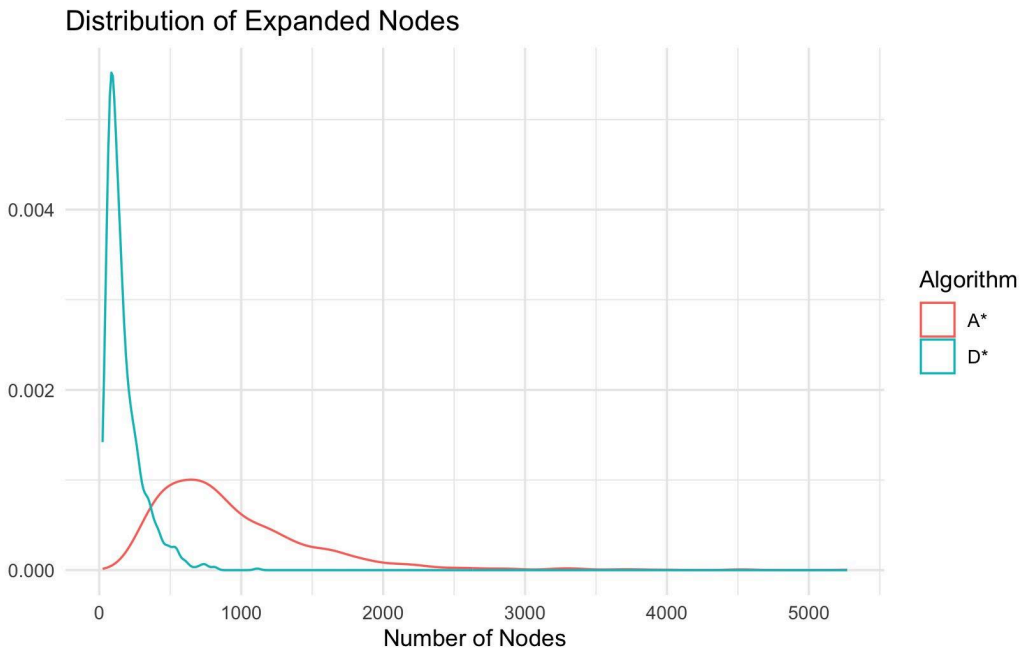

Figure 11: Distribution of expanded nodes

Figure 11 clearly shows that on average D* algorithm explores significantly fewer nodes than A*. The difference is huge, with D* expanding 170 nodes compared to 930 nodes for A*. This big difference is attributed to the fact that D* efficiently rescues the path steps it has computed before encountering a new obstacle instead of recalculating the path in each step. This property of updating the path only if it's necessary makes D* more efficient in dynamically changing mazes.

4.4 Conclusion

In this paper, different approaches for finding a path - A*, Greedy, and D* - in the dynamic maze were discussed and analyzed both separately and comparatively. To assess each algorithm's strengths and limitations, we considered numerous factors, such as an average number of steps and computational cost under varying conditions.

The analyses have shown that A* and D* perform better in changing environments, performing fewer steps in different-sized mazes. Here, A* significantly outperforms D*. On the other hand, when the maze is entirely changing, The Greedy Local search works best, making the least number of steps. When it comes to the mean number of expanded nodes, D* outperforms A* due to its property of recalculation path only when it encounters an obstacle. Considering all these findings, we concluded that the choice of the algorithm depends on the specific requirements of the environment. In the initially defined maze, D* is the most preferable one as it shows less computational cost, and it performs reasonably well in terms of the average number of steps taken. In the future, further optimizations and the integration of hybrid approaches could be found that combine the strengths of each algorithm.

References

[Koe01] [Likh01] Sven Koenig. Maxim Likhachev. D* Lite, 2001.
[Har68] [Nill68] [Ber68] Peter Hart, Nils Nilsson, and Raphael Bertram. A Formal Basis for the Heuristic Determination of Minimum Cost Paths, 1968.
[Koe]. [Likh]. [Har68]. [Nill68]. [Ber68].
Path, I. (2018). *Advanced 1. Incremental Path Planning*. [online] YouTube. Available at: https://youtu.be/_4u9W1xOuts?si=rnCj91Ce8S1bLy1A [Accessed 17 Nov. 2024].
School, W. (2024). What School Didn't Tell You About Mazes #SoMEpi. [online] YouTube. Available at: https://youtu.be/uctN47p_KVk?si=YfQUewoJ5BG3J0qG [Accessed 8 Dec. 2024].