GROUP SECTION

COMPARATIVE ANALYSIS OF SEARCH ALGORITHMS IN A DUNGEON EXPLORER ENVIRONMENT

A. Introduction:

This report evaluates the performance of various search algorithms-Breadth-First Search (BFS), Depth-First Search (DFS), Uniform Cost Search (UCS), and A*-in navigating a grid-based maze in a Dungeon Explorer Environment where a robotic agent is tasked with finding the path to a designated target. The report focuses on the efficiency of all algorithms in finding paths, computational costs, and their adaptability to different configurations of the environment.

B. Robot and Environment Setup:

The Dungeon Explorer is a custom environment designed to evaluate the performance of search algorithms in dynamic and challenging maze scenarios.

ROBOT: Visualized as a human-like figure with a circular shape, navigates the dungeon for exploration, moving in the four cardinal directions (up, down, left, and right). Each movement incurs a cost of 1, as illustrated in the *figure 1* below.

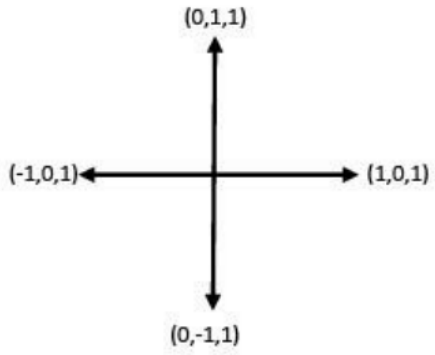


Figure 1: Direction of the agent

ENVIRONMENT: The Grid-Based Maze is represented as a grid with cells, where traps and blocks are randomly distributed according to specified parameters such as density and spacing.

GRID ELEMENTS: **S**: Entry Cell (Green), G: Exit cell (red), *: Free space, #: Obstacles (Black), X: Traps or impassable cells.

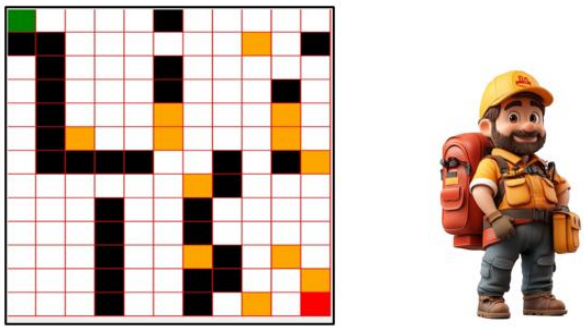


Figure 2: Dungeon Environment and the explorer

GRID DIMENSIONS: 13x11 cells.

TRAP: +1 COST

BLOCK: OBSTACLE

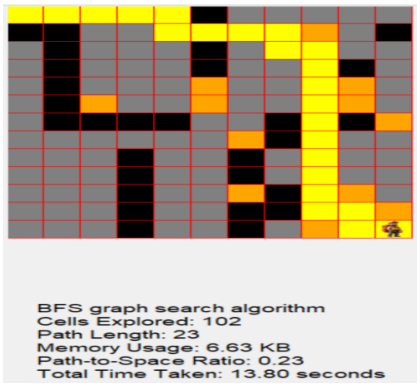
INITIAL STATE: (0,0)

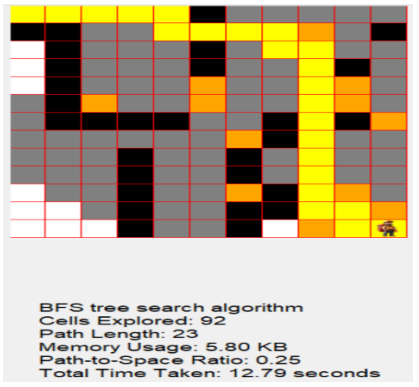
GOAL STATE: (12,10)

KEY PARAMETER ANALYSED: Cells Explored, Memory Usage, Path-to-Spacing Ratio, Path Length, Total Time Taken to get out of the dungeon.

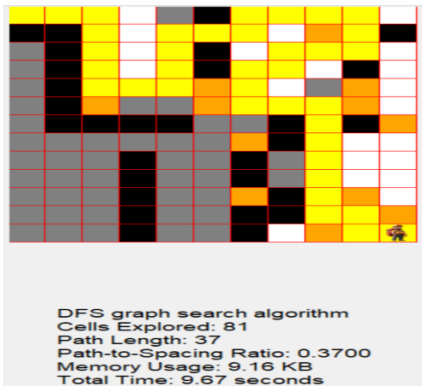
C. Experiment Result:

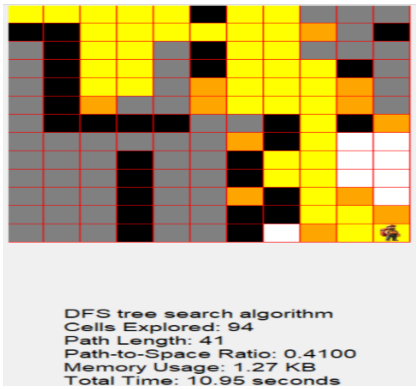
BFS:



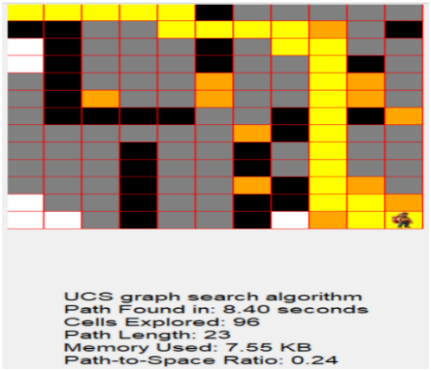


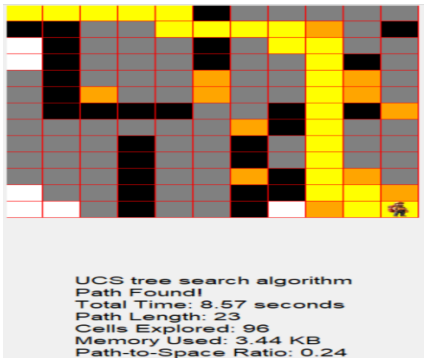
DFS:



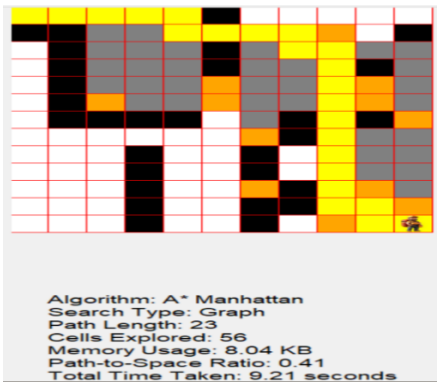


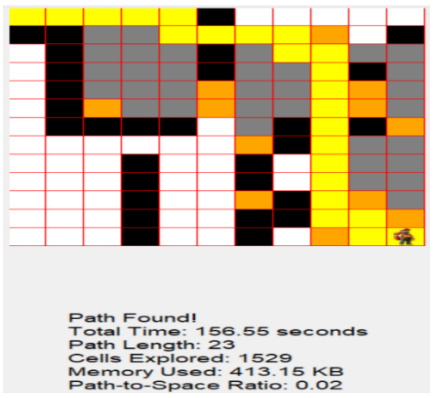
UCS:





A*:





D. Path Explored:

The table below represents the paths generated by different search algorithms—BFS, DFS, UCS, and A*-in both graph and tree-based dungeon environment. From this table, its inferred that BFS, UCS, A* consistently finds the shortest paths.

BFS Graph	[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4), (1, 5), (1, 6), (1, 7), (2, 7), (2, 8), (3, 8), (4, 8), (5, 8), (6, 8), (7, 8), (8, 8), (9, 8), (10, 8), (11, 8), (11, 9), (12, 9), (12, 10)]
BFS Tree	[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4), (1, 5), (1, 6), (1, 7), (2, 7), (2, 8), (3, 8), (4, 8), (5, 8), (6, 8), (7, 8), (8, 8), (9, 8), (10, 8), (11, 8), (11, 9), (12, 9), (12, 10)]
DFS Graph	[(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (3, 2), (4, 2), (4, 3), (4, 4), (3, 4), (2, 4), (1, 4), (1, 5), (1, 6), (0, 6), (0, 7), (0, 8), (0, 9), (1, 9), (2, 9), (2, 8), (2, 7), (3, 7), (3, 6), (4, 6), (5, 6), (5, 7), (5, 8), (6, 8), (7, 8), (8, 8), (9, 8), (10, 8), (11, 8), (11, 9), (12, 9), (12, 10)]
DFS Tree	[(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (3, 2), (4, 2), (4, 3), (3, 3), (2, 3), (1, 3), (0, 3), (0, 4), (1, 4), (1, 5), (1, 6), (0, 6), (0, 7), (1, 7), (2, 7), (2, 6), (3, 6), (4, 6), (5, 6), (5, 7), (4, 7), (3, 7), (3, 8), (4, 8), (5, 8), (6, 8), (7, 8), (8, 8), (8, 7), (9, 7), (9, 8), (10, 8), (11, 8), (11, 9), (12, 9), (12, 10)]
UCS Graph	[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4), (1, 5), (1, 6), (1, 7), (2, 7), (2, 8), (3, 8), (4, 8), (5, 8), (6, 8), (7, 8), (8, 8), (9, 8), (10, 8), (11, 8), (11, 9), (12, 9), (12, 10)]
UCS Tree	[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4), (1, 5), (1, 6), (1, 7), (2, 7), (2, 8), (3, 8), (4, 8), (5, 8), (6, 8), (7, 8), (8, 8), (9, 8), (10, 8), (11, 8), (11, 9), (12, 9), (12, 10)]
A* Graph	[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4), (1, 5), (1, 6), (1, 7), (2, 7), (2, 8), (3, 8), (4, 8), (5, 8), (6, 8), (7, 8), (8, 8), (9, 8), (10, 8), (11, 8), (11, 9), (12, 9), (12, 10)]
A* Tree	[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4), (1, 5), (1, 6), (1, 7), (2, 7), (2, 8), (3, 8), (4, 8), (5, 8), (6, 8), (7, 8), (8, 8), (9, 8), (10, 8), (11, 8), (11, 9), (12, 9), (12, 10)]

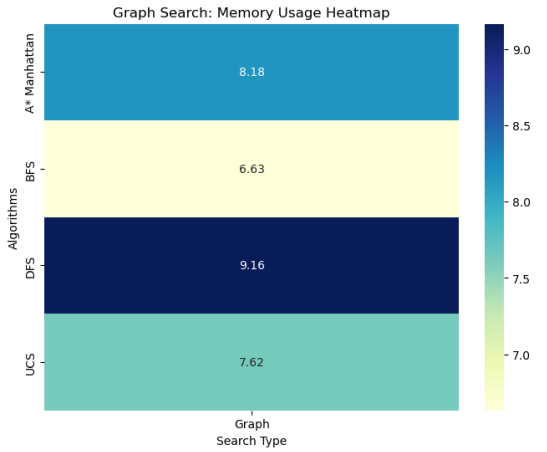
E. Evaluation Table:

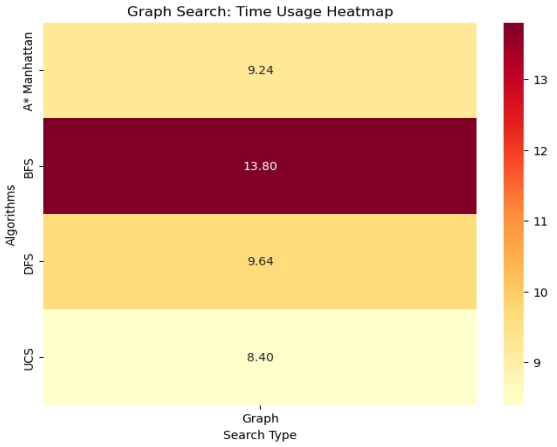
Methods	Search Algorithms	Path Length	Cells Explored	Memory (Kb)	Time (Seconds)	Path to Space Ratio
GRAPH SEARCH	BFS	23	102	6.63	13.80	0.23
	DFS	37	81	9.16	9.67	0.37
	UCS	23	96	7.55	8.40	0.24
	A*Manhattan	23	56	8.04	9.21	0.41
TREE SEARCH	BFS	23	92	5.80	12.79	0.25
	DFS	41	94	1.27	10.95	0.41
	UCS	23	96	3.44	8.57	0.24
	A*Manhattan	23	1529	413.15	156.55	0.02

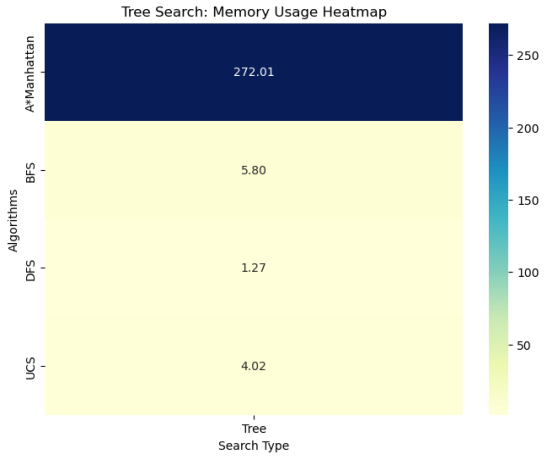
F. Charts of results:

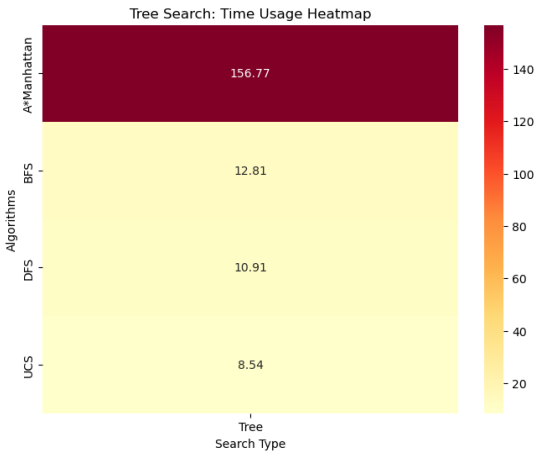
In addition to the table of results, we compare the time and space complexity of the algorithms using heatmaps below. Heatmaps were specifically chosen for this analysis because both memory usage and time are relatively low in scale and do not translate effectively when represented in conventional graphs. By using heatmaps, the intensity and range of memory and time used by the algorithms can be visually analysed through colour gradients.

The heatmaps below effectively highlight the differences and allow for intuitive interpretation of these complexities.









G. Analysis:

Graph Search:

From the evaluation, BFS, UCS, and A* (with the Manhattan heuristic function) yielded shortest paths, with an estimated path length close to the shortest possible one, while DFS found the longest paths. UCS proved optimal as it favors paths with the lowest total cost. BFS also ensures the shortest path when all edge costs are uniform, guaranteeing optimality, especially with an admissible heuristic, making A* a reliable choice. However, DFS doesn't always guarantee the most optimal solution but at times can be more efficient cause of deep search.

In the memory usage heatmap, A* showed higher memory requirements (8.18 KB) due to storing nodes and heuristic evaluations. BFS and UCS also exhibited considerable memory usage (6.63 KB and 7.62 KB, respectively), attributed to their node storage mechanisms. DFS uses very least memory (9.16 KB), when

memory is limited, it will be effective. The time usage heatmap revealed that BFS took the longest time to execute (13.80 seconds), whereas UCS and A* completed faster at 8.40 and 9.24 seconds, respectively. DFS, despite its lack of guarantees, showed slightly faster execution at 9.67 seconds. This indicates that the choice of the heuristic and tree structure significantly impacts algorithm performance.

Tree Search:

From the set of tree search algorithms, the algorithm that performed better is referred to as Uniform Cost Search (UCS). It performed best with a path length of 23 and explored only 96 cells, far fewer than DFS and A* using the Manhattan heuristic. UCS was highly efficient with memory, using just (3.44 KB), attributed to its systematic exploration and avoidance of revisiting nodes. Despite exploring 94 cells, DFS only used 1.27 KB of memory, making it ideal for low-memory environments. A* using the Manhattan heuristic consumed a massive 413.15 KB of memory and took 156.55 seconds to complete, which aligns with its exhaustive exploration approach in non-optimized environments.

Cross Comparison: Graph vs. Tree Search:

When comparing graph search with tree search, graph search clearly outperformed tree search across various metrics. BFS in graph search explored fewer cells (102 versus 92) and executed faster (13.80 versus 12.79 seconds) due to avoiding revisiting nodes. Similarly, graph-based UCS and DFS demonstrated better memory usage compared to their tree search counterparts. A significant difference emerged with A* using the Manhattan heuristic: graph search examined only 56 cells and used 8.04 KB of memory, while tree search explored 1,529 cells and required 413.15 KB. This clear difference shows graph search is efficient in designing the algorithms to eliminate redundant computation.

Best Algorithm:

A* with the Manhattan distance heuristic in the graph search paradigm is clearly the top-performing algorithm across all methods. It finds the shortest path with a length of 23, explored only 56 cells, and completes the task in a quick 9.21 seconds. Thanks to its heuristic design, A* focuses on the most promising paths first, making it especially effective in environments with obstacles. Its combination of efficiency, speed, and accuracy makes it the best choice among all algorithms in both tree and graph search approaches.

F. Discussion and Conclusion:

When solving real-world problems, it's important to choose an algorithm that fits the problem's needs, like how the data is organized and the limits on memory and time. The graph-based A* algorithm is a great choice because it's flexible and finds the best path, even in changing environments. In the future, algorithms can be improved with smarter rules and real-time feedback to better adapt to changes.

INDIVIDUAL SECTION

PATH FINDING OPTIMIZATION IN AUTONOMOUS DRIVING SYSTEM USING SEARCH ALGORITHMS

1.INTRODUCTION TO AI

1.1. History and Evolution of AI:

AI is a cluster of technology where system able to perform certain tasks without the human brain. AI aims to perform multiple functions like learning, solving and decision making. The journey of Artificial Intelligence begun in the year 1930 by Alan Turing and Godel, made a foundation for logical computational systems [6]. In 1950, the introduction of logical Theorist and LISP language at Dartmouth Conference, is considered as the birth of AI [6]. This paves the way for decades of innovation. Despite the limited computational power available to simulate brains, McCulloch and Hebb's, invention of neural networks in 1940, paved a way for development of algorithms capable of logical reasoning and the data-driven learning [6]. Today's AI poses vast technologies, such DNN, DQL, NLP and robotics. Modern Autonomous systems, such as IBM's Watson and Google's AlphaGo, demonstrate the field's sophisticated capabilities in areas ranging from playing complex games to driving cars autonomously and diseases diagnosis.

2.CURRENT STATE OF AI IN VARIOUS DOMAINS

2.1. Human-Robot Interaction:

Currently, AI has made significant advancement in the Human-robot Interaction (HRI) Field through the application of technologies like – NLP, Computer Vision, Machine learning enabling robots to adapt and interact to human activities and speech more accurately, leading to more dynamic interaction [13]. Reinforcement learning have also taken this flexibility a level higher as robots improve their actions corresponding to what the users or the environment gives in return. Human emotions are now recognized by socially intelligent robots with emergent emotion recognition application hence creating new opportunities in healthcare, education and companionship [13].

2.2 Deep Learning:

In this era, Deep learning is making innovative changes and modifications through all aspects of our live [8]. The Deep Learning methodology depicts the human brain with deep neural networks. One of the best examples of deep learning is AlphaGo, a program known for defeating a professional human -Go- player, which was considered to be one of the "grand challenges of AI" [8]. Deep learning has proven to be able to break barriers in front of AI. Nowadays in healthcare, deep learning algorithms assist in diagnosing diseases from MRI surpassing human experts. It is also transforming the autonomous industry by developing autonomous vehicles.

2.3. Robotics (Autonomous Systems):

The current state of AI in robotics is marked by rapid advancements that are evolving across growing technologies. In Autonomous systems, AI enabled greater adaptability for dynamic environment [7]. Social robots are companion and human friendly robots that operate collaboratively with their human counterparts in areas such as manufacturing, health care and military. Ai driven autonomous robots are good at decision making can perform certain risky tasks like obstacle avoidance, collision detection and path navigation based on the dynamic environment [7].

3.THE APPLICATIONS OF SEARCH ALGORITHMS IN ROBOTICS

Search algorithm is used to find the optimal path/direction to reach goal from starting point, considering various obstacles. In Robotics, search algorithms can be applied to finding near optimal solutions for decision-making problems [14]. These algorithms enable robots to adjust their actions dynamically from the learnt experience, leading to more effective problem-solving capabilities [14]. Applications ranges from robot pathfinding to pick items in warehouse to advanced scenarios like automated vehicles navigating city streets.

3.1. Autonomous Driving System-Path Finding issue:

Autonomous Driving systems (ADSs) refers to self-driving vehicles, which are designed to navigate routes and make continues decisions the aligns with the traffic rules and constraints programmed in them [9]. Utilising combinations of Sensors/Cameras, radars, PID controller and artificially intelligent software, driving systems perceive surroundings, interpret visual information and plan optimal routes independently [9]. ADS rely significantly on search algorithms and AI for path planning and navigation. Finding efficient paths while avoiding collisions in real-time and across complex environments is a key challenge [9]. Search techniques like A* and Dijkstra's, sampling-based methods like RRT*, Probabilistic RoadMap (PRM) and heuristic approaches are commonly used to address these issues. The industrial applications of ADS include logistics, autonomous public transportation, and on-demand ride-sharing services, showcasing the potential of AI to revolutionize mobility while ensuring safety and efficiency.

This report proceeds to survey state of art search techniques researched for path-planning in ADSs, analysing capabilities on key metrices, their pros and cons and concluding the suggestion for direction of improvement.

4. GRAPH SEARCH ALGORITHMS IN PATHFINDING

4.1. A* algorithm:

The A* algorithm is a heuristic based path finding algorithm, established to find a shortest optimal path between any two points in graph or grid. The **Rationale** behind this algorithm is to address the need for efficient path finding in dynamic terrains environments. This algorithm utilizes the efficiency of Dijkstra's algorithm to search for all the possible paths and make sure it chooses the shortest one, with the data obtaining feature of Greedy Best-First Search by moving towards the most probable paths at the quickest rate. It operates by expanding paths from the starting node, using two lists (open and closed) to manage nodes pending exploration or already evaluated [4]. Nodes are selected based on

$$f(n)=g(n)+h(n)$$

where, g(n) denotes the actual path value from initial to current node and h(n) is the heuristic cost.

Algorithm 1: A* Algorithm

Assign x_{init} as *open*
Calculate $\hat{f}(x_{init})$ from the start point x_{init} to all possible *open* nodes n_i
Find the smallest $\hat{f}(x_{init})$ and select the associated node n
If $\hat{f}(x_{init})$ is *empty*
    No path Exist
End
While $n \notin x_{goal}$
    Node n is marked *closed*
    Apply the successor operator Γ
    The $\hat{f}$ function is re–calculated for successor nodes of n marking these nodes as *open*
    If $\hat{f}$ is *empty*
        No path Exist
    Else If $\hat{f}(n_i)$ is now smaller than when n_i was *closed*
        Successor closed nodes of n are remarked as *open*
    End
End
Parent node n is *closed* (Path Found)[26]

In the above figure [11], A* continues to explore nodes by expanding the selected node, calculating the costs for neighbouring nodes, until the goal is reached, or no nodes remain to be explored.

In Robotics, A* helps in real time path planning, as quickly adapts to dynamic environment changes and other Obstacles. The paper [4] also explores the improvements made in combining A* with other obstacle avoidance techniques to design a compound navigation technique including the Dynamic Window Approach (DWS). This combination of algorithms applies A* for global path planning and DWA for local obstacle avoidance to provide better, efficient control of robot navigation in complex environments as well as in changing surroundings and conditions.

4.2. Dijkstra Algorithm:

This algorithm was introduced by Edsger W. Dijkstra in 1956, it's a graph-based approach to find the lest distance path between the nodes, which are connected by weighted edges [5]. It starts by placing the source node in the tentative list with an initial distance of zero. Then consecutively moves the smallest distance nodes to the permanent list and updates the distances of its neighbouring nodes. This process continues until the tentative list is empty, indicating that the shortest paths to all reachable nodes have been established [5]. This Algorithm is used in many areas especially routing.

In Robotics, this algorithm is used in order to select the safe and optimal paths for ADS, based on avoidable intersections and minimal travel distance and fuel consumption [1]. The **rationale** behind the Dijkstra Algorithm; its reliable methodology to find the shortest deterministic path in weighted graphs. It operates to determine the lowest cost path at any distance, making its application where pathfinding is crucial with time constraints. As discussed in [1], Dijkstra's algorithm (*Figure 1*) is utilized to solve a shortest path problem in a directed graph by identifying an optimal sub-graph Q that minimizes the distance from a source vertex s to a target vertex t. It calculates the path's distance using the sum of edge weights between consecutive vertices in Q.

Algorithm 1: *Dijkstra_Shortest_Path* (G, s, t)

// Inputs: An oriented diagraph G, a source and target vertices (s, t).
// Outputs: The shortest path from s to t and the length of such path.

1. Initialize the current vertex c to s and the distance between the current vertex and every vertex v in the graph to infinity: $d(v) \leftarrow \infty$

2. Initialize the optimal path set of vertices to the source vertex s : $Q = \{s\}$.

3. Update the distance of the vertices adjacent to the current vertex as:

$$d(v) \leftarrow \min(d(v), d(c) + W(c, v)) \quad (5)$$

where $W(c, v)$ is the weight of the edge (c, v).

4. Compare the distances of those vertices to the current vertex.

5. Move to the vertex with the smallest distance and mark it as the current vertex.

6. Include this vertex in the optimal set of vertices, Q.

7. Repeat steps **3-6** until reaching the target vertex.

8. Output the optimal set of vertices and the sum of the consecutive weights from the source to the target.

Figure 3 Algorithm

this algorithm significantly enhances the efficiency and safety of navigational tasks in robotics.

5. STATE-OF-ART ALGORITHMS IN ADS PATH FINDING

Advanced Algorithms like RRT* and PRM tackles the limitations of graph-search methods by randomly sampling the continuous state space. This strategy bypasses the discretization problems associated with more traditional techniques, enabling a broader and more flexible exploration of possible pathways.

5.1 Rapidly- Exploring RoadMap Tree Star Algorithm (RRT*):

RRT* is a variant of RRT algorithm which was introduced by Steven M. LaValle and James J. Kuffner in 1998 [12]. RRT is a foundation sampling-based algorithm for path finding, that efficiently explores high-dimensional spaces by incrementally growing a tree. It guarantees feasibility

While RRT can handle complex environments, path generated by RRT was not optimal. The **Rationale** behind RRT* is mainly to address this issue in ADS. The RRT* was proposed by Karaman and Frazzoli in 2011, with the most efficient asymmetrical property [12]. RRT* improved path quality by introducing two major features:

- tree rewiring
- Least cost parent selection

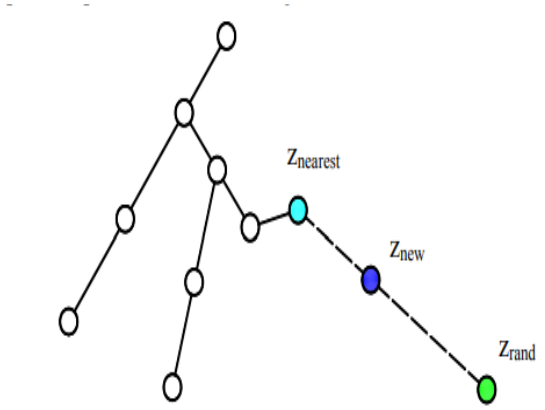
These features enhance path quality through optimal parent selection and efficient tree restructuring.

Algorithm 1. $T = (V, E) \leftarrow RRT^*(z_{init})$

```
T ← InitializeTree();
T ← InsertNode(Ø, z_init, T);
for i=0 to i=N do
    z_rand ← Sample(i);
    z_nearest ← Nearest(T, z_rand);
    (z_new, U_new) ← Steer (z_nearest, z_rand);
    if Obstaclefree(z_new) then
        z_near ← Near(T, z_new, |V|);
        z_min ← Chooseparent (z_near, z_nearest, z_new);
        T ← InsertNode(z_min, z_new, T);
        T ← Rewire (T, z_near, z_min, z_new);
return T
```

Figure 4

In the *figure 2*, During incremental growth of a tree, **RRT*** searches for nearest nodes in the existing tree and connects them to newly sampled points. This Steer ((*Znearest, Zrand*) method involves identifying the optimal path to connect the newly sampled space to the existing tree, allowing efficient navigation [12].



In pathfinding for ADS, safe and efficient navigation is critical. RRT* constructs a tree like structure in the space (Z), which includes obstacle-free regions (*Zfree*) and goal states (*Zgoal*) The algorithm ensures that the tree grows toward the goal while avoiding obstacles and optimizing the path.

In the give figure**,** A random point (*Zrand*) is selected, and the nearest existing node (*Znearest*) is determined. If the distance is within the preset step size, a new node (*Znew*) is added to the tree. A collision check ensures that the path between *Znearest* and *Znear* lies entirely within *Zfree* . This process enables incremental tree growth in the obstacle-free region, balancing exploration and path optimization [12]. Several **improvements** are made to the standard RRT*, such as Bidirectional RRT*, Informed RRT*, RRT-Connect*[12]. These advancements incorporate good features of the RRT* algorithm with much superior convergence processes.

5.1.a. Issues and mitigation strategies for RRT*:

Main limitations with the RRT* algorithm is post-processing requirements, such as pruning and smoothing, are essential for creating optimal paths, especially for non-holonomic robots, which struggle with kinematic constraints [12]. To mitigate these issues, in [12] author proposes, smoothing methods such as Bézier curves and B-splines ensure optimal paths with continuous curvature. Modified RRT* variants incorporate dynamic re-planning to address non-holonomic constraints, improving path feasibility for vehicles and robots in dynamic environments. These strategies collectively enhance RRT*'s robustness and efficiency, making it more suitable for pathfinding challenges in ADS.

5.2 Probabilistic Road Map Algorithm (PRM):

PRM had emerged in 1990 by Lydia E. Kavraki, is a sampling-based motion planning algorithm, which constructs a graph by selecting nodes in the configuration space in a stochastic manner and connects the nodes

by finding collision free paths [3]. In real time the roadmaps serve as a graph, where nodes are the feasible states and edges represents the transition between states. PRM consist of two phases [10]:

- **Learning Phase:** robot motions are randomly sampled, with each node assessed for nearby neighbours to build a comprehensive roadmap.
- **Query Phase:** It finds the shortest path between the start and end nodes of the constructed road map using algorithms like A* and Dijkstra.

Algorithm 1 PRM

1: $V \leftarrow \emptyset$
2: $E \leftarrow \emptyset$
3: loop
4: $q \leftarrow$ a randomly chosen free configuration
5: $V_q \leftarrow$ a set of candidate neighbors of q chosen from V
6: $V \leftarrow V \{q\}$
7: *for* $q' \in V_q$, in order of increasing $D(q, q')$, *do*
8: *if* E = same_connected_component$(q, q') < |(q, q')|$ *then*
9: $E \leftarrow E \cup \{(q, q')\}$
10: update E's connected components

Figure 5

In the Learning Phase, Graph is constructed by making it suitable for static or minimally dynamic environments. *Figure3* shows the code structure of Unidirectional graph (V, E) which V are sampled points and E denotes valid edges between them.
This constructed graph is used in Querying phase to connect the start position and goal position to nearby points in the graph to finds the shortest path. This PRM algorithm efficiently handles complex environments by simplifying the search for a path into a graph search problem, reducing computational complexity [3]. In ADS, it plays an essential role in the navigation systems by providing a reliable method to compute feasible paths, avoid obstacles, and ultimately guide the vehicle towards its destination in an optimized manner. The **rationale** behind the PRM algorithm is to address the difficulty of path finding in high dimensional and volatile environment. Conventional path planning algorithms had shortcomings of scalability and efficiency when it was required to find paths in vast spaces filled with numerous obstacles [3]. PRM solves this problem by using a sampling-based technique which constructs a graph of possible motions in the C-space.

5.2.a. Issues and mitigation strategies for PRM:

Though PRM is effective in high-dimensional spaces, its performance can be affected by issues such as low sampling in narrow passages, redundant paths, and high computational cost when working with large environments [10]. To overcome these limitations, Bidirectional PRM (PRM*) [10] and hybrid PRM, in combination with other algorithms like RRT and Artificial Potential Fields (APF), have been shown to improve its efficiency and robustness, particularly in cluttered or dynamically changing environments.

6. PROS AND CONS OF SEARCH ALGORITHMS

As inferred from [4] [1] [12] [10], The table below provides pros and cons of all four algorithms.

Algorithm	Pros	Cons
A*	Guarantees shortest Path with consistent heuristic approach. Gives deterministic results.	Due to its heuristic approach, it offers no guarantee of success even if a path exists. Inefficient in dynamic environments
Dijkstra	Guarantees shortest path in graph	Time consuming Inefficient for high-dimensional space
RRT*	can handle a large class of non-holonomic dynamical systems exhibits asymptotic optimality	Requires large memory and it prioritizes feasibility over cost-efficiency.
PRM	High dimensional space can be handled well	Optimality is not guaranteed Fails in complex dynamic environment

7. GUIDELINES AND ETHICS OF AI

The growth and development of AI should adhere to the guidelines and ethics to ensure safe and responsible applications [2]. Ethical AI development is guided by frameworks from organizations like IEEE and UNESCO, focusing on transparency, accountability, privacy and security, which sets the guidelines for AI application to behave ethically and responsibly, ensuring that it aligns with social values and promotes the safe and fair use of technology [2]. Main goal as a part of inclusive development is to make sure that AI incorporates value by appreciating the inputs of the stakeholders.

8. CONCLUSION

AI in autonomous driving is a fantastic technological and innovation combination. AI might alter transportation mobility, provide a safe and more efficient and equitable solution despite difficulties. AI's potential is autonomous driving is huge and increasing as we enter a new transportation age. Advanced AI technology integration has equipped ADS to securely analyze the environmental data necessary for them to manage unpredictable contexts, obstacles, and perform tasks independently. These systems rely on sophisticated algorithms for perception, decision making and decision implementing that are designed to make efficient operation possible with the least human intervention [2]. Autonomous vehicles must be designed, produced, and deployed in strict adherence to explicit regulations and guidelines. This involves tackling critical issues such as accountability, data confidentiality, and the ethical implications of algorithmic decision-making, particularly in high-stakes scenarios.

9. REFERENCES

[1] Alshammrei, S., Boubaker, S. and Kolsi, L., 2022. Improved Dijkstra algorithm for mobile robot path planning and obstacle avoidance. *Comput. Mater. Contin*, *72*, pp.5939-5954.

[2] Atakishiyev, S., Salameh, M., Yao, H. and Goebel, R., 2024. Explainable artificial intelligence for autonomous driving: A comprehensive overview and field guide for future research directions. *IEEE Access*.

[3] Chen, J., Zhou, Y., Gong, J. and Deng, Y., 2019, July. An improved probabilistic roadmap algorithm with potential field function for path planning of quadrotor. In *2019 Chinese control conference (CCC)* (pp. 3248-3253). IEEE.

[4] Chen, Y., Pang, J., Gou, Y., Lin, Z., Zheng, S. and Chen, D., 2024. Research on the A* Algorithm for Automatic Guided Vehicles in Large-Scale Maps. *Applied Sciences*, 14(22), p.10097. Available at: https://doi.org/10.3390/app142210097.

[5] Dhulkefl, E., Durdu, A. and Terzioğlu, H., 2020. Dijkstra algorithm using UAV path planning. *Konya Journal of Engineering Sciences*, *8*, pp.92-105.

[6] Ertel, W., 2024. *Introduction to artificial intelligence*. Springer Nature.

[7] Hagos, D.H. and Rawat, D.B., 2022. Recent advances in artificial intelligence and tactical autonomy: Current status, challenges, and perspectives. *Sensors*, 22(24), p.9916. Available at: https://doi.org/10.3390/s22249916

[8] Kelleher, J.D., 2019. *Deep learning*. MIT press.

[9] Langbauer, D., Path-Finding Algorithm for Autonomous Robot-Car.

[10] Li, Q., Xu, Y., Bu, S. and Yang, J., 2022. Smart vehicle path planning based on modified PRM algorithm. *Sensors*, *22*(17), p.6581.

[11] Muhammad, A., Abdullah, N.R.H., Ali, M.A., Shanono, I.H. and Samad, R., 2022, May. Simulation performance comparison of A*, GLS, RRT and PRM path planning algorithms. In *2022 IEEE 12th Symposium on Computer Applications & Industrial Electronics (ISCAIE)* (pp. 258-263). IEEE.

[12] Noreen, I., Khan, A. and Habib, Z., 2016. Optimal path planning using RRT* based approaches: a survey and future directions. *International Journal of Advanced Computer Science and Applications*, *7*(11).

[13] Obaigbena, A., Lottu, O.A., Ugwuanyi, E.D., Jacks, B.S., Sodiya, E.O. and Daraojimba, O.D., 2024. AI and human-robot interaction: A review of recent advances and challenges. *GSC Advanced Research and Reviews*, *18*(2), pp.321-330.

[14] Senanayake, M., Senthooran, I., Barca, J.C., Chung, H., Kamruzzaman, J. and Murshed, M., 2016. Search and tracking algorithms for swarms of robots: A survey. *Robotics and Autonomous Systems*, *75*, pp.422-434.