NATIONAL ECONOMICS UNIVERSITY, HANOI
COLLEGE OF TECHNOLOGY
FACULTY OF DATA SCIENCE AND ARTIFICIAL INTELLIGENCE

INTRODUCTION TO AI (TOKT11121.AI66A)

REPORT

Topic: Maze Pathfinding

Instructor: Nguyen Trong Nghia

Student: Mai Huy Dang - 11247269

Mai Tuan Manh - 1124738

Nguyen Huy Minh - 11247322

Hanoi, October 2025

1 Introduction

Emergencies such as fires, earthquakes, or terrorist incidents in crowded environments pose significant threats to human life. In these situations, ensuring a rapid and safe evacuation becomes a critical concern for event organizers and public safety authorities. A single delay or misjudgment in finding the nearest exit can result in severe injuries or fatalities. As modern events like museum exhibitions, stadium concerts, or conferences often attract thousands of participants, the design of intelligent evacuation systems capable of identifying optimal escape routes has become an important research area in **Artificial Intelligence (AI).**

Evacuation planning can be modeled as a **state space search problem**, where individuals navigate through a structured environment—such as a building or maze—to reach the nearest safe exit. Each possible position corresponds to a *state*, and moving from one point to another constitutes a *state transition*. The challenge lies in determining the sequence of movements that minimizes both time and distance, while adapting to dynamic conditions such as congestion or blocked paths. By representing this scenario as a **maze pathfinding problem**, AI search algorithms can be systematically analyzed and compared in a controlled environment before being applied to real-world contexts.

In this study, we focus on three fundamental search algorithms that represent distinct approaches to problem-solving in AI:

- **Breadth-First Search (BFS)**—a systematic, exhaustive algorithm that explores all possible paths level by level to guarantee the discovery of the *shortest possible route*.

- **Depth-First Search (DFS)**—a recursive strategy that explores deeply along one direction before backtracking, which can quickly find a path but does not ensure optimality.

- **Greedy Best-First Search (GBFS)**—a heuristic-based method that uses an *evaluation function* (in this case, Manhattan distance) to estimate how close a node is to the goal, prioritizing speed over accuracy.

These algorithms are applied to a simulated **maze environment** that represents a simplified model of a real building layout, such as a conference hall or exhibition area.

By using randomly generated mazes of varying complexity, the study evaluates algorithm performance under different spatial constraints. Metrics such as **runtime**, **memory consumption**, and **path optimality** are used to assess each algorithm's strengths and weaknesses.

Beyond theoretical comparison, this research emphasizes **practical implications**. For example, in a large museum like the A80 or a stadium with multiple exits, an algorithm that can quickly generate near-optimal escape routes may be more valuable than one that guarantees perfect accuracy but is too slow to respond in real time. Thus, the trade-off between *optimality* and *efficiency* is central to this analysis.

Furthermore, this work forms the foundation for developing **agent-based evacuation simulations (ABS)**, where each individual is modeled as an intelligent agent capable of independent decision-making. Integrating search algorithms like BFS and GBFS into such simulations allows researchers to study collective evacuation behavior, congestion dynamics, and the impact of environmental variables such as smoke or blocked exits.

In summary, the goals of this project are:

- To model evacuation planning as a maze-based search problem.
- To apply and compare BFS, DFS, and GBFS algorithms across various maze configurations.
- To evaluate algorithmic performance based on efficiency, optimality, and resource consumption.
- To derive insights that can guide real-world evacuation planning and AI-driven safety systems.

Through this comparative analysis, we aim to bridge the gap between theoretical search algorithms and their real-world applications in **emergency management** and **intelligent building systems**.

2 Problem Formulation

To represent real-world evacuation dynamics, the environment is modeled as a **two-dimensional maze**. Each cell corresponds to a possible position that an individual can occupy.

- **Start position (S):** Represents a person's initial location.
- **Walls (#):** Represent obstacles, booths, or inaccessible areas.
- **Exits (E):** Represent safe escape points.

The goal is to find the **shortest** and **safest path** from *S* to the nearest *E* while avoiding obstacles.

2.1 State Representation

- Each state defines a position (x, y) in the maze grid.
- Successor states are reachable by moving up, down, left, or right.
- The transition model defines legal moves that do not cross walls.
- The cost of each move is uniform (1 unit).

2.2 Heuristic Function for GBFS

Since no dynamic cost data is provided (e.g., smoke or congestion), GBFS employs the **Manhattan distance** as the heuristic:

$$|| x_a - x_b ||_M = | x_{a1} - x_{b1} | + | x_{a2} - x_{b2} |$$

This heuristic estimates the distance from the current position to the nearest exit and guides the search efficiently toward the goal.

3 Algorithms

3.1 Maze Generation

The maze serves as a simplified representation of a real-world environment such as a building floor plan, exhibition hall, or stadium. To construct the maze, we employ the **DFS backtracking algorithm**, which generates what is known as a **perfect maze**—a structure where there exists *exactly one unique path* between any two open cells. The algorithm operates by recursively visiting cells, carving pathways, and backtracking when necessary until all cells are accessible.

However, perfectly generated mazes are often too structured and predictable. To better simulate real-world evacuation scenarios, where multiple alternative routes and intersections exist, we introduce **controlled randomness** by selectively removing a small proportion of walls after the initial generation. This process results in an **imperfect maze**, containing **loops, branching corridors, and multiple exits**. The inclusion of these irregularities mirror realistic conditions such as alternate doors, interconnected rooms, and unexpected detours.

Each maze is stored as a **two-dimensional grid matrix**, where:

- . represents a traversable cell.
- # represents a wall or obstacle.
- S marks the **starting position** (the initial location of a person or agent).
- E denotes one or more **exit points** (the safe destinations).

This structure facilitates flexible adaptation to different algorithms, allowing the same environment to be explored using BFS, DFS, and GBFS for direct performance comparison.

We adapted the Python code for Perfect Mazes from the dataset available at Kaggle. The original code was modified and extended by adding new features developed by us.

3.2 Search Methods

Three classical search algorithms are implemented to navigate through the maze from the starting point to the nearest exit.

Breadth-First Search (BFS)

```
# Breadth-First Search
from collections import deque

def bfs(grid):
    start, exits, rows, cols = parse_maze(grid)
    q = deque([start])
    parent = {}
    visited = {start}
    while q:
```

```
        cur = q.popleft()
        if cur in exits:
            return reconstruct_path(parent, cur)
        for nb in neighbors(cur, rows, cols, grid):
            if nb not in visited:
                visited.add(nb)
                parent[nb] = cur
                q.append(nb)
    return None
```

BFS explores the maze **level by level**, visiting all nodes at the current depth before expanding deeper levels. It employs a **queue data structure**, ensuring that the first discovered path to an exit is also the **shortest possible path**. While BFS guarantees optimality, it requires maintaining a record of all visited nodes and frontier nodes, leading to **high memory consumption**, especially in large mazes. Despite this limitation, BFS serves as a reliable benchmark for evaluating other algorithms.

Depth-First Search (DFS)

```
# Depth-First Search
def dfs(grid):
    start, exits, rows, cols = parse_maze(grid)
    stack = [start]
    parent = {}
    visited = {start}
    while stack:
        cur = stack.pop()
        if cur in exits:
            return reconstruct_path(parent, cur)
        for nb in neighbors(cur, rows, cols, grid):
            if nb not in visited:
                visited.add(nb)
                parent[nb] = cur
                stack.append(nb)
    return None
```

DFS explores as far as possible along one direction before backtracking. It utilizes a **stack structure** (or recursion) to keep track of exploration. While DFS may quickly reach a goal if it happens to choose the correct path early, this does not guarantee finding the shortest route. In dense or looping mazes, DFS can become inefficient, revisiting the same regions multiple times and consuming unnecessary computation. Nevertheless, its

simplicity and low per-step overhead make it an important algorithm for baseline comparison.

Greedy Best-First Search (GBFS)

GBFS introduces a **heuristic-based evaluation function**, prioritizing nodes that appear closest to the goal according to an estimated cost function $h(n)$. In this study, we use the **Manhattan Distance heuristic**:

$$|| x_a - x_b ||_M = | x_{a1} - x_{b1} | + | x_{a2} - x_{b2} |$$

```python
# Greedy Best-First Search
import heapq

def manhattan(a, b):
    return abs(a[0]-b[0]) + abs(a[1]-b[1])

def gbfs(grid):
    start, exits, rows, cols = parse_maze(grid)
    def h(node):
        return min(manhattan(node, e) for e in exits)
    heap = [(h(start), start)]
    parent = {}
    visited = {start}
    while heap:
        _, cur = heapq.heappop(heap)
        if cur in exits:
            return reconstruct_path(parent, cur)
        for nb in neighbors(cur, rows, cols, grid):
            if nb not in visited:
                visited.add(nb)
                parent[nb] = cur
                heapq.heappush(heap, (h(nb), nb))
    return None
```

The algorithm selects the next node to expand based on the lowest heuristic value, aiming to move rapidly toward the goal. GBFS can significantly reduce search time and memory usage, but heuristic inaccuracies may mislead it, leading to suboptimal paths or local minima. However, its **speed and efficiency** render it appropriate for real-time applications like emergency navigation or robot pathfinding.

3.3 Benchmarking and Evaluation Metrics

To ensure fair and consistent performance assessment, we implement a **benchmarking function** that executes each algorithm on multiple randomly generated maze instances. The following evaluation metrics are recorded:

- **Average runtime (seconds)**—measures computational efficiency.
- **Peak memory usage (bytes)**—indicates the space complexity of each algorithm.
- **Path optimality (steps)**—represents how close the discovered route is to the true shortest path.

This multi-metric analysis enables both quantitative and qualitative comparison, highlighting the trade-offs between computational cost, accuracy, and responsiveness.

4 Experiment and Results

4.1 Experimental Setup

Experiments are conducted across a series of mazes with increasing dimensions—**5×5**, **7×7**, **51×51**, and **201×201** grids—to assess algorithm scalability. Each maze includes **one starting point (S)** and **multiple exits (E)** strategically positioned at the boundaries to mimic real building layouts. For each maze size:

- 100 random maze instances are generated using the DFS backtracking method.
- Each algorithm (BFS, DFS, GBFS) is executed **100 times** on distinct maze configurations.
- The **average runtime**, **peak memory consumption**, and **path length** are computed.

All algorithms are implemented in Python using standardized data structures to ensure consistency. Timing and memory measurements are captured using built-in profiling tools. Each experiment is repeated multiple times to reduce random variability and measurement noise.

4.2 Performance Analysis

Small-scale Mazes (5×5 and 7x7)

```
#5x5
Benchmark results over 100 mazes (5x5, 2 exits):

BFS: avg runtime 0.000351s, avg peak memory 2.23 KB,
avg path length 3.64
DFS: avg runtime 0.000378s, avg peak memory 1.73 KB,
avg path length 5.96
GBFS: avg runtime 0.000380s, avg peak memory 2.03 KB,
avg path length 3.64

#7x7
Benchmark results over 100 mazes (7x7, 2 exits):

BFS: avg runtime 0.000599s, avg peak memory 2.61 KB,
avg path length 6.94
DFS: avg runtime 0.000659s, avg peak memory 2.69 KB,
avg path length 11.67
GBFS: avg runtime 0.000610s, avg peak memory 2.23 KB,
avg path length 6.95
```

At small maze sizes, all algorithms complete successfully in negligible time (below 1 ms on average). BFS consistently produces the **shortest paths**, as expected, while DFS and GBFS achieve similar performance with minor deviations in path length. In such limited environments, memory and runtime differences are statistically insignificant, demonstrating that the choice of algorithm has little practical effect for small evacuation areas.

Medium-scale Mazes (51×51)

```
#51x51
Benchmark results over 100 mazes (51x51, 4 exits):

BFS: avg runtime 0.009952s, avg peak memory 31.46 KB,
avg path length 140.28
DFS: avg runtime 0.011884s, avg peak memory 56.58 KB,
avg path length 362.92
GBFS: avg runtime 0.010097s, avg peak memory 22.41 KB,
avg path length 144.02
```

As maze complexity increases, the performance divergence becomes more apparent. BFS continues to guarantee the shortest route but exhibits **noticeable memory overhead** due to its exhaustive search strategy. DFS begins to display variability depending on the maze layout; in certain cases, it finds a route quickly, while in others, it becomes trapped in long dead-end paths. GBFS shows the **best balance** at this scale—completing searches faster than BFS while maintaining near-optimal path quality.

Large-scale Mazes (201×201)

```
#201x201
Benchmark results over 100 mazes (201x201, 10 exits):

BFS: avg runtime 0.179215s, avg peak memory 182.08 KB,
avg path length 564.80
DFS: avg runtime 0.211539s, avg peak memory 618.96 KB,
avg path length 2985.14
GBFS: avg runtime 0.192165s, avg peak memory 101.64 KB,
avg path length 592.63
```

In large and complex mazes, the performance gap becomes substantial:

- **BFS** maintains accuracy but consumes exponentially higher memory, making it less feasible for real-time systems.
- **DFS** becomes unstable and inefficient, often revisiting large sections of the maze and occasionally failing to terminate within reasonable time limits.
- **GBFS**, while designed to be faster by using the Manhattan heuristic to guide the search, can sometimes become slower when multiple exits are present, since it must repeatedly compute heuristic values for each possible goal. (With a smarter and faster heuristic than Manhattan, GBFS could perform even better)

4.3 Summary of findings

The comparative results demonstrate that each algorithm offers distinct advantages depending on the situation:

Algorithm	Path Optimality	Runtime Efficiency	Memory Usage	Suitability
BFS	Always optimal	Slow in large mazes	High	Safety-critical planning
DFS	Non-optimal	Variable	Moderate	Exploratory analysis only
GBFS	Sub-optimal but close	Fastest overall	Low	Real-time decision making

In real-world applications, **BFS** is preferred when the shortest path is essential (e.g., pre-planned evacuation routes). **GBFS** is ideal for **time-sensitive operations**, such as live crowd management or robotic guidance. **DFS**, while less efficient, remains a useful educational and analytical tool for understanding search behavior.

5 Discussion

The comparative analysis of BFS, DFS, and GBFS across multiple maze configurations highlights both the theoretical and practical implications of each algorithm in evacuation pathfinding. While the experiments focus on abstract maze environments, the insights derived can be effectively extended to **real-world evacuation systems**, particularly in **large-scale public events** such as stadiums, museums, or conference venues.

5.1 Algorithmic Insights and Trade-offs

Each search algorithm exhibits unique characteristics that influence its suitability for emergency evacuation applications.

- **Breadth-First Search** ensures that the path discovered is always **optimal** in terms of the fewest number of moves or shortest distance to an exit. This makes BFS a reliable choice for situations where **accuracy and safety** take precedence over computation time. However, the cost of such optimality is **exponential growth in**

memory usage and processing time, particularly in large environments with many open pathways. Thus, BFS is most appropriate for **offline evacuation planning**, such as verifying building layouts or precomputing emergency routes before an event takes place.

- **Depth-First Search**, due to its simple recursive structure, requires relatively low setup overhead and minimal auxiliary data structures. However, its tendency to pursue deep branches before backtracking often leads to inefficient exploration, especially in dense or looping mazes. In evacuation contexts, DFS could result in **long detours or dead ends**, making it unreliable for real-time decision systems. Nevertheless, it remains valuable as a **baseline algorithm** for educational or prototype purposes, illustrating the importance of exploration order in search problems.

- **Greedy Best-First Search** introduces a heuristic-driven approach that provides a strong trade-off between speed and optimality. By prioritizing nodes with smaller Manhattan distances to the nearest exit, GBFS can produce solutions **faster** than BFS, while still maintaining near-optimal path quality. This balance makes GBFS particularly well-suited for **dynamic evacuation systems**, **autonomous robots**, or **intelligent guidance systems** where rapid computation is essential and small deviations from the optimal path are acceptable.

5.2 Real-world Applications and Implementation Scenarios

The insights gained from maze-based pathfinding extend naturally to **evacuation modeling** in complex infrastructures. In practical terms, these algorithms can be embedded in digital systems for:

1. **Evacuation Route Optimization:** Identifying the shortest or fastest paths to exits for crowd control systems in real time.

2. **Guidance Robots and Smart Displays:** Providing adaptive navigation instructions in venues using GBFS or hybrid algorithms (e.g., A*).

3. **Safety Policy Simulation:** Allowing planners to test various evacuation layouts under simulated crowd conditions before events occur.

Moreover, **hybrid methods** combining BFS's completeness with GBFS's efficiency (such as the A* algorithm) could further improve the balance between speed and accuracy. These methods can integrate additional dynamic factors such as **crowd density**, **smoke propagation**, or **blocked exits** to better reflect real emergency conditions.

5.3 Case Study: Evacuation Simulation in the A80 Stadium

To demonstrate the practical applicability of the algorithms, we conducted a **simulation experiment** based on the **A80 Stadium**, a large event venue frequently hosting thousands of spectators. The A80 layout was represented as a grid-based map with **five main exits** and multiple internal pathways connecting seating areas, corridors, and open spaces. Each person inside the stadium was modeled as an **autonomous agent**, each with independent movement, perception, and decision-making capabilities.

Simulation Design

- Each **agent** begins from a random location within the stadium, representing a spectator's position at the onset of an emergency.
- **Walls and booths** are modeled as obstacles, and **exits** correspond to doors leading outside the venue.
- Each agent applies one of the tested algorithms—BFS, DFS, or GBFS—to find the nearest available exit.
- Agents move simultaneously, and congestion is simulated: when too many agents occupy the same cell or corridor, their **movement speed decreases** proportionally to local crowd density.
- Agents continuously update their chosen routes; if congestion near an exit becomes severe, they can **recalculate and switch** to an alternative exit.

Insights and Implications

The A80 case study shows that how well an algorithm works in theory has a real effect on how well it works in practice during an evacuation. While BFS remains the gold standard for route optimality, **GBFS emerges as the most pragmatic choice** for real-time evacuation management due to its balance between computational efficiency and practical effectiveness. In dynamic, unpredictable environments, the ability to react swiftly often outweighs the need for mathematical perfection.

These findings suggest that integrating GBFS-based pathfinding into **agent-based simulation platforms** and **smart building management systems** can significantly improve emergency response efficiency. Moreover, hybrid methods such as **A***—combining BFS's optimal guarantees with GBFS's heuristic speed—hold promise for next-generation evacuation technologies.

6 Conclusion

This study explored the application of classical search algorithms—**Breadth-First Search**, **Depth-First Search**, and **Greedy Best-First Search**—to model and solve the **evacuation pathfinding problem** in complex environments. By representing evacuation routes as maze-based search tasks, we systematically analyzed the trade-offs between computational cost, optimality, and real-world usability.

The experimental results across mazes of varying complexity clearly demonstrate distinct algorithmic characteristics:

- **BFS** produces the shortest and safest paths but is computationally expensive.
- **DFS** is highly dependent on path direction and often inefficient.
- **GBFS**, guided by Manhattan distance, provides the best balance between speed and practicality.

Future Works

While this study provides foundational insights, several directions remain open for future research:

- **Hybrid Search Methods:** Combining BFS's completeness with GBFS's speed
- (as in **A*** or **Weighted A*** algorithms) to achieve optimality and efficiency simultaneously.
- **Dynamic Heuristics:** Incorporating real-time factors—such as heat, smoke, and human density—into heuristic functions for more realistic modeling.
- **Parallel and Distributed Search:** Leveraging multi-core or cloud computing to handle large-scale, multi-agent evacuation simulations efficiently.
- **Machine Learning–Enhanced Heuristics:** Training neural networks to learn heuristic estimates from simulation data, enabling adaptive and intelligent decision-making.
- **Real-World Implementation:** Integrating these algorithms into IoT-enabled smart buildings or emergency response systems for field testing and validation.

Final Remarks

In conclusion, this research demonstrates how fundamental AI search algorithms can be adapted and evaluated for **life-critical applications** such as evacuation planning. Among the algorithms studied, **GBFS emerges as the most practical solution**, offering a strong balance between computational speed and route effectiveness. The **A80 case study** further illustrates how heuristic-guided strategies can reduce evacuation times and improve safety outcomes in large-scale events.

By bridging the gap between **theoretical AI models** and **real-world safety systems**, this study contributes to the growing field of **intelligent evacuation planning**—where artificial intelligence not only enhances efficiency but also helps protect human lives in times of crisis.