Hercules’ Journey to his Wifey

Project Report

American University of Armenia, CSE

CS 246/346 Artificial Intelligence

Anzhelika Simonyan

Armen Mkrtumyan

Zara Grigoryan

FALL 2024

Abstract

This paper explores the development of an interactive AI-driven game inspired by the mythological Labors of Hercules. The game models a dynamic search environment where Hercules, as an intelligent agent, must navigate a grid-like world to reach a goal, overcoming obstacles such as walls, lava, and the Lernaean Hydra. The agent utilizes a variety of search algorithms, including Breadth-First Search (BFS), Depth-First Search (DFS), Uniform-Cost Search (UCS), and A* search. By introducing uncertainty and probabilistic reasoning in the form of the Hydra monster, the game transitions from a deterministic environment to a stochastic one. This paper presents a comparative analysis of the search strategies, evaluates their effectiveness under various grid configurations, and discusses how logical reasoning and constraint satisfaction can be incorporated into Hercules' world to enhance decision-making.

Keywords: Artificial Intelligence, Pathfinding Algorithms, Grid-based Navigation, A* Algorithm, Probabilistic Reasoning, Agent-based Systems, AI Game Design

List of Contents

1. List of Figures

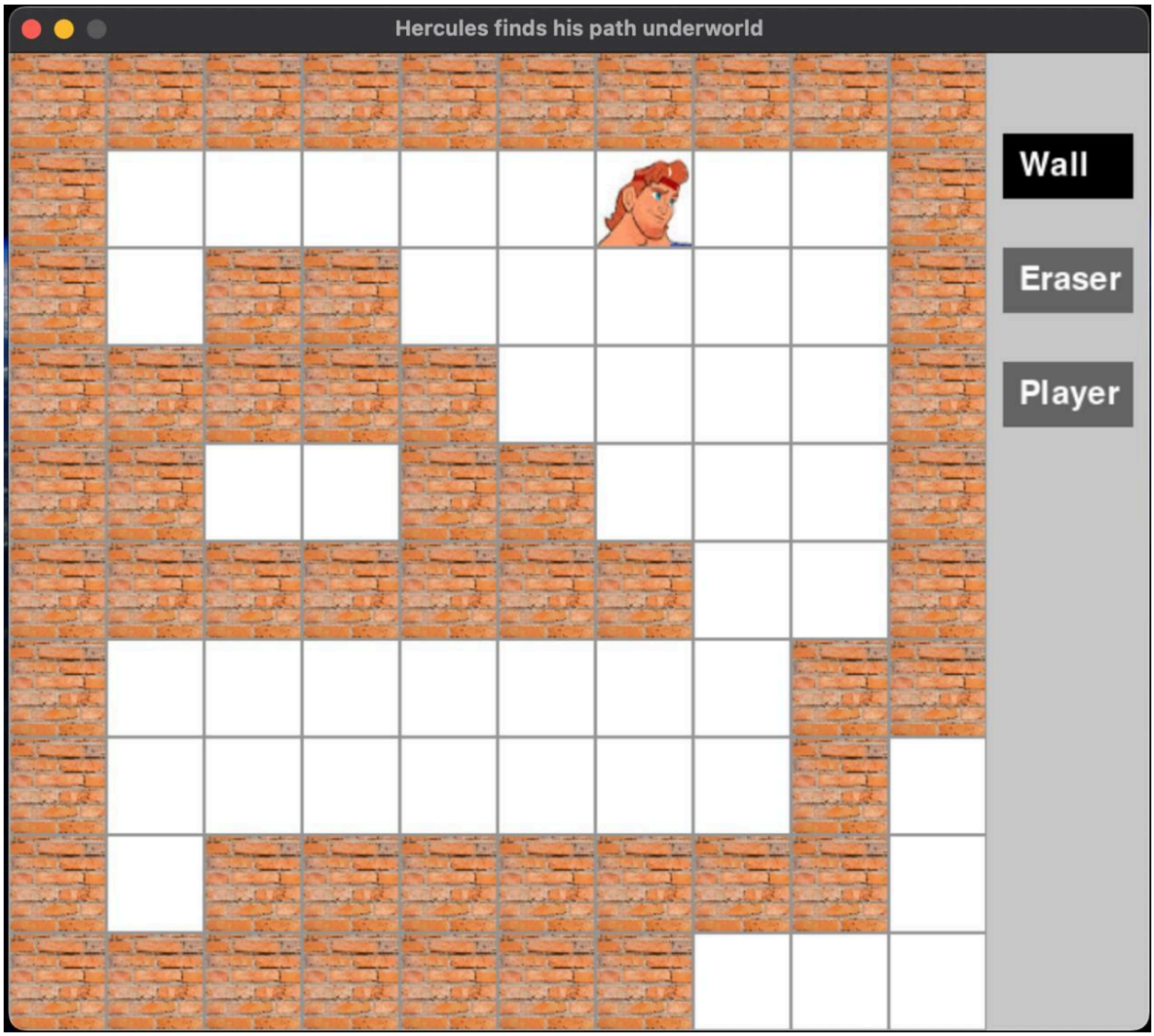
2. Introduction

“A hero is someone who has given his or her life to something bigger than oneself” - Joseph Campbell

Hercules, driven to madness, committed a terrible crime against his own family. Overcome with guilt, he sought a path to redemption. To atone, he was assigned twelve seemingly impossible tasks, each presenting immense challenges and formidable adversaries, known as the Twelve Labors of Hercules [TED-Ed, 2018].

Inspired by this myth, we set out to create a game that brings Hercules’ legendary journey to life. Our project features Hercules as an intelligent agent navigating a series of challenges modeled after the Twelve Labors. Through this, we aim to explore the capabilities of artificial intelligence in solving constrained problems and pathfinding tasks in dynamically created environments.

3. Problem definition



The project begins with defining the agent's state as a user-defined grid-like environment. Users place objects like walls (blocking access), lava, mountains, a start, and a goal square. The grid size and obstacle layout are flexible, allowing users to control difficulty. Randomly generated maps are used for calculations, with the Hydra monster optionally included. Hercules navigates this world to reach the goal, overcoming obstacles and constraints inspired by mythological tasks.

3.1. Search Strategies and Their Applications

In the context of this project, the grid can have a wide variety of configurations. The map can range from a simple open grid with only a Hercules and a goal to a complex maze with walls, obstacles, and dynamic monsters representing Hercules' mythical adversaries (however, in the scope of this project, we are not considering monsters as dynamic agents who can also make decisions, it is left out). This variability necessitates the use of powerful search strategies to solve the navigation problem efficiently. We consider both uninformed and informed search strategies. BFS and DFS find a path for the relaxed problem, that is we assume the obstacles are not placed on the map. [See section 6]

3.1.1. Breadth-First Search (BFS)

The first search strategy we considered in the project is breadth-first search (BFS). Breadth-First Search (BFS) systematically explores adjacent squares to find the shallowest path to the solution. Starting from the agent's position $(x,\ y)$, BFS examines squares in the order:
$(x-1,\ y),\ (x+1,\ y), (x,\ y-1), (x,\ y+1)$ - corresponding to left, right, down, and up movements. The algorithm's time and space complexities are $O(b^d)$, where b is the branching factor and d is the depth of the shallowest solution. In this project, b ranges from 2 (corner positions) to 4 (central positions). As an uninformed search strategy, BFS does not account for obstacles like lava, mountains, or monsters. BFS solves the relaxed problem - obstacles (except for walls) can be passed through and energy points do not count. Its main strength is finding shallow solutions, but it becomes inefficient in deeper or highly constrained grids due to exponential growth in time and memory usage.

3.1.2. Depth-First Search (DFS)

The next search strategy, which is again uninformed, is depth-first search(DFS). In general case, running time of DFS is exponential - $O(b^m)$, where m is the maximum depth of the grid. Space complexity is linear - $O(bm)$, as DFS requires storing only the current path. It should be noted that the maximum depth of the grid m is the same as the size of the grid. The algorithm effectively explores adjacent squares in Down, Up, Right, Left order:
$((x,\ y-1),\ (x,\ y+1), (x+1,\ y), (x-1,\ y))$. The applicability of DFS is useful for exploring deep paths, but it is not ideal for finding the shortest path. Because DFS uses a LIFO queue, it tends to get stuck in deep, irrelevant branches without constraints. As our tests seem to indicate, on average DFS is the worst among the other search algorithms in terms of the running times [See section 6]. Like BFS, DFS finds a path for the relaxed problem.

3.1.3. Uniform-Cost Search (UCS, aka Dijsktra for large graphs)

Uniform-Cost Search (UCS) accounts for obstacles like lava, mountains, and monsters by using a cost function g to calculate the path cost. The agent starts with a fixed amount of energy, which decreases by 1 point for each move (up, down, left, or right). Moving onto a mountain decreases energy by 10 points, while stepping on lava or encountering a monster results in failure. UCS prioritizes paths with lower costs, avoiding mountains if an alternative path requires fewer than 10 moves. Lava and monsters are avoided unless blocking the only path to the goal. Unlike DFS and BFS, UCS uses empirical analysis for performance evaluation rather than theoretical complexity. For UCS, as well as A*, energy points are calculated as n^2, where n is the size of the grid.

3.1.4. A* search algorithm

Unlike the rest of the algorithms discussed above, A* search algorithm is an informed strategy, allowing us to use a heuristic function. A* combines path cost so far (g) and estimated cost to the goal (h). The key advantage of using A* search algorithm is that having a good heuristic, meaning a heuristic that is admissible and consistent, we obtain optimal and complete solutions. In this project we have chosen Manhattan distance as the heuristic function for A* search, which is both consistent and admissible, making the search algorithm both optimal and complete. The next section uncovers Manhattan distance heuristic in more detail. More efficient in practice with a good heuristic, worst case like bfs. More tests done.

3.1.5. Local Search

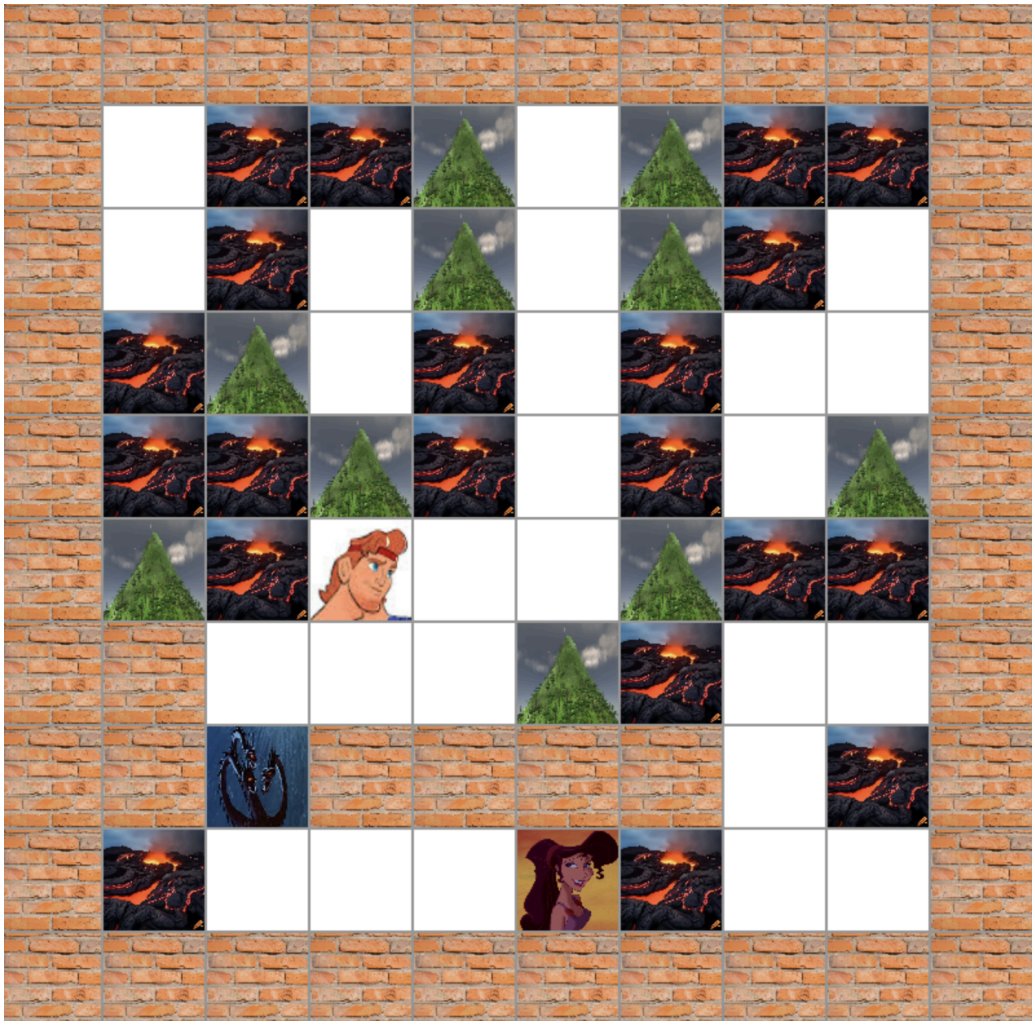
In the context of this project, local search algorithms were explored as a means of navigating the dynamic and obstacle-filled grid environment. Local search, specifically hill climbing, operates by iteratively improving the agent's position based on heuristic evaluations, such as minimizing the Manhattan distance to the goal. To address its susceptibility to getting trapped in local optima, random restarts were incorporated, allowing the search to retry from different initial states. However, given the stochastic nature of the grid, including dynamic obstacles like Hydra and lava, local search faces significant challenges, as evidenced by an average success rate of 25.72% across 50 test runs [See section 6]. This highlights the inherent difficulty of applying local search in unknown, probabilistic environments.

3.2. Lernaean Hydra - **the dreadful monster**

Inspired by the myth, a Hydra monster was included to introduce stochasticity. According to legend, cutting off a Hydra's head causes two more to grow. This concept was applied using probabilities, transforming the environment from deterministic to stochastic. The monster is

randomly placed on the grid. While BFS and DFS solve a relaxed problem ignoring obstacles, UCS and A* treat the Hydra as an obstacle. Unlike walls, however, the Hydra can be killed with probability p. If the Hydra blocks the only path to the goal, Hercules must attempt to kill it, requiring proximity and sufficient energy, adding uncertainty to the search process.

Assume we have a similar instance of the map as shown below:



Here, the goal square is enclosed with walls and lava squares, and while BFS and DFS will ignore the lava square, A* and UCS search will find a path passing through the square that contains Hydra. Without the functionality of killing, the algorithms will not find solutions, with the functionality of killing they will find it. We can calculate different probabilities and try to estimate how many tries the monster will be killed with some confidence level. The number of tries n is also dependent on the energy level of the agent. We first calculate the probability of killing the monster, that is the same as hitting the right head, in n trials.

Let m_k denote the number of heads after the k-th attempt. Initially, before the agent hits any of the heads, $m_0 = 3$. The probability of hitting the right head is defined as follows: $p_k = \frac{1}{m_k}$.

If the sequence of random variables $X_1, X_2, \ldots, X_k$ represent the head chosen at the k-th trial, then $X_1, X_2, \ldots, X_k \sim DU(\frac{1}{m_k})$. After each missed attempt, Hydra grows two more heads: $m_{k+1} = m_k + 1$, and since initially $m_0 = 3$, $m_k = 3 + k$. We calculate the probability of hitting the right head by first calculating the probability of missing the right head in n trials.

$$P(All\ Misses) = \prod_{k=0}^{n-1}(1 - \frac{1}{3+k}) = \frac{\prod_{k=0}^{n-1}(2+k)}{\prod_{k=0}^{n-1}(3+k)} = \frac{\frac{(1+n)!}{1!}}{\frac{(2+n)!}{2!}} = \frac{2}{n+2}$$

This probability converges to 0, indicating that for a large number of trials the chances of missing all of them are in fact quite small: $\frac{2}{n+2}$.

Then, the probability that in n trials agent will actually kill the Hydra is:

$$P(n) = 1 - \frac{2}{n+2} = \frac{n}{n+2} \to 1$$

Now we can make more calculations based on this probability. For example, we can estimate with 95% probability that with a given energy level E, the agent will kill the Hydra in 18 trials. Assume each hit costs 10 energy points. Then, n will be the number of attempts the agent can make. Since each attempt costs 10 units of energy, the number of attempts is: $n = \lfloor \frac{E}{10} \rfloor$. We need to check if the agent can kill the Hydra with 95% confidence: $P(n) \geq 0.95$. If this inequality holds, then with 95% confidence, the agent will kill the Hydra using n attempts.

$$P(n) = \frac{n}{n+2}$$

$$\frac{n}{n+2} \geq 0.95$$

$$n \geq 0.95(n + 2)$$

And then it is trivial to see $n \geq 18$, meaning $E \geq 180$ in order to kill the Hydra with 95% probability.

4. Literature Review/Search

In this section, we will first demonstrate that the Manhattan distance heuristic is both admissible and consistent, ensuring that the A* algorithm remains optimal and complete. Next, we will explore the Chebyshev distance heuristic and its applicability in grid environments that permit movement in eight directions, including diagonals. Finally, we will discuss advancements in pathfinding algorithms, focusing on the "Relaxed Dijkstra and A* with linear complexity" approach for large-scale grid environments.

4.1. Admissibility and Consistency of the Manhattan Distance Heuristic

The Manhattan distance heuristic calculates the sum of the absolute differences between the current position and the goal position along each coordinate axis. Formally, for a point (x_1, y_1) and a goal (x_2, y_2), the Manhattan distance is defined as: $h(n) = |x_2 - x_1| + |y_2 - y_1|$

Admissibility: A heuristic is admissible if it never overestimates the true cost to reach the goal. In grid environments where movement is restricted to horizontal and vertical steps, the Manhattan distance represents the minimum number of moves required to reach the goal, assuming no obstacles. Therefore, it does not overestimate the actual cost, making it an admissible heuristic.

Consistency: A heuristic is consistent (or monotonic) if, for every node n and each successor n', the estimated cost $h(n)$ is less than or equal to the cost of reaching n' plus the estimated cost from n' to the goal:

$h(n) \leq c(n, n') + h(n')$

In the context of the Manhattan distance, this inequality holds because moving from n to n' typically involves a unit step, and the heuristic difference aligns with this step cost. Therefore, the Manhattan distance heuristic is consistent [Patel, Heuristics].

Since the Manhattan distance heuristic is both admissible and consistent, the A* algorithm using this heuristic is guaranteed to find the optimal path and is complete. For example, in the N-puzzle, $g(n)$ counts the number of moves made so far, and $h(n)$ can be the Manhattan distance—adding up how many spaces each tile is from where it belongs. This gives a good sense of progress without overestimating the moves needed, ensuring A* works optimally. However, A* has its challenges. It can end up using a lot of memory because it keeps track of many partially explored paths, which makes it harder to use in larger or more complex scenarios. To address this, Korf (1985) came up with an improvement called Iterative Deepening A*. This version reduces the memory needed by only keeping track of what's absolutely necessary, while still guaranteeing the best solution [TR91-16].

4.2. Chebyshev Distance in Grid Environments with Diagonal Movement

In grid environments that allow movement in eight directions (horizontal, vertical, and diagonal), the Chebyshev distance heuristic becomes relevant. The Chebyshev distance between two points (x_1, y_1) and $(x_2, y_2)$is defined as:

$$h(n) = max(|x_2 - x_1|, |y_2 - y_1|)$$

This metric reflects the minimum number of moves a king would take in chess to move from one square to another, considering it can move one square in any direction. In pathfinding, this heuristic accurately estimates the shortest path in scenarios where diagonal movements are allowed and have the same cost as orthogonal movements. Therefore, the Chebyshev distance serves as an admissible and consistent heuristic in such grid environments, ensuring that A* remains optimal and complete [Chugani, 2024].

4.3. Relaxed Dijkstra and A Algorithms for Large-Scale Grid Environments*

Traditional pathfinding algorithms like Dijkstra's and A* can become computationally intensive in large-scale grids with numerous obstacles. To address this, Ammar et al. (2014) introduced the Relaxed Dijkstra (RD) and Relaxed A* (RA*) algorithms, which exploit the structural properties of grid environments to achieve linear time complexity.

The RD algorithm simplifies the computation of the exact cost function g by computing it at most once for each cell during the entire search. This approach reduces the processing time required to find the (near)-optimal solution. Simulation results indicate that RD consistently provides the optimal path in 100% of cases for four-neighbor grids and over 97% for eight-neighbor grids.

Similarly, the RA* algorithm relaxes the exact cost computation, allowing for faster pathfinding while maintaining near-optimal solutions. The primary advantage of these approaches is their ability to process both four- and eight-neighbor grid environments efficiently, making them suitable for large-scale pathfinding problems.

In summary, the Manhattan and Chebyshev distance heuristics are effective in their respective grid environments, ensuring that A* search remains both optimal and complete. For large-scale grids with extensive obstacles, the Relaxed Dijkstra and A* algorithms offer promising solutions

with linear complexity, enhancing computational efficiency without significantly compromising path optimality.

5. Methodology

The development of the interactive AI-driven game *Hercules' Journey to His Wifey* was guided by a structured methodology that emphasized the integration of various AI techniques and search algorithms. This approach aimed to create an intelligent agent, Hercules, who could navigate a dynamically generated grid-like environment filled with obstacles, such as walls, lava, and the Lernaean Hydra, to reach a specified goal.

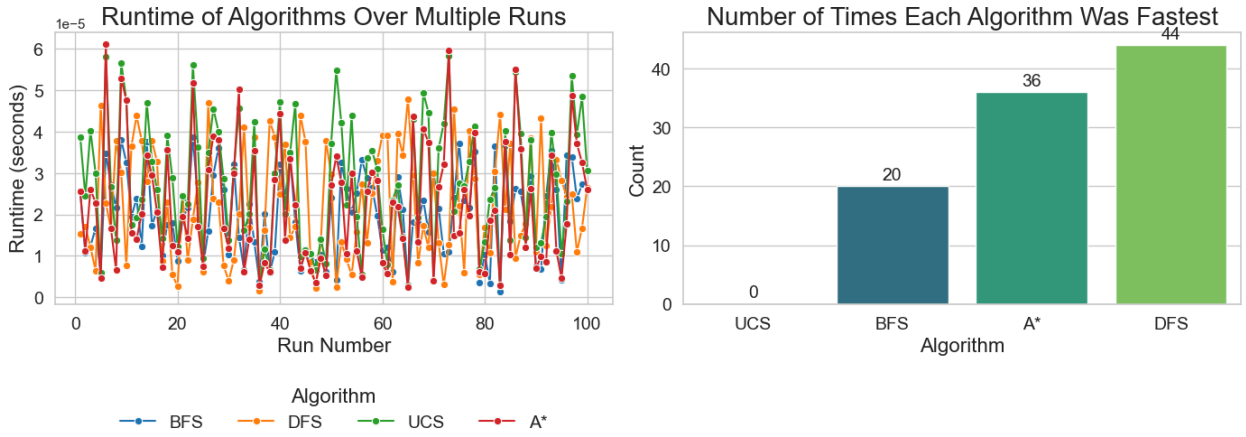
The methodology begins with defining Hercules' environment as a customizable grid where users place objects like walls, lava, mountains, the start, and goal squares. The grid's layout and size are flexible, allowing users to control the environment's complexity. Hercules' objective is to navigate from start to goal while overcoming obstacles and minimizing energy use. To introduce stochasticity, a Hydra monster was added, transforming the problem from deterministic to stochastic.

To navigate this environment, search algorithms were implemented, including BFS, DFS, UCS, and A* Search, each with unique strengths and weaknesses. Hercules can only move to adjacent squares unless blocked by walls, lava, or the Hydra, with formal constraints ensuring accessibility rules are upheld.

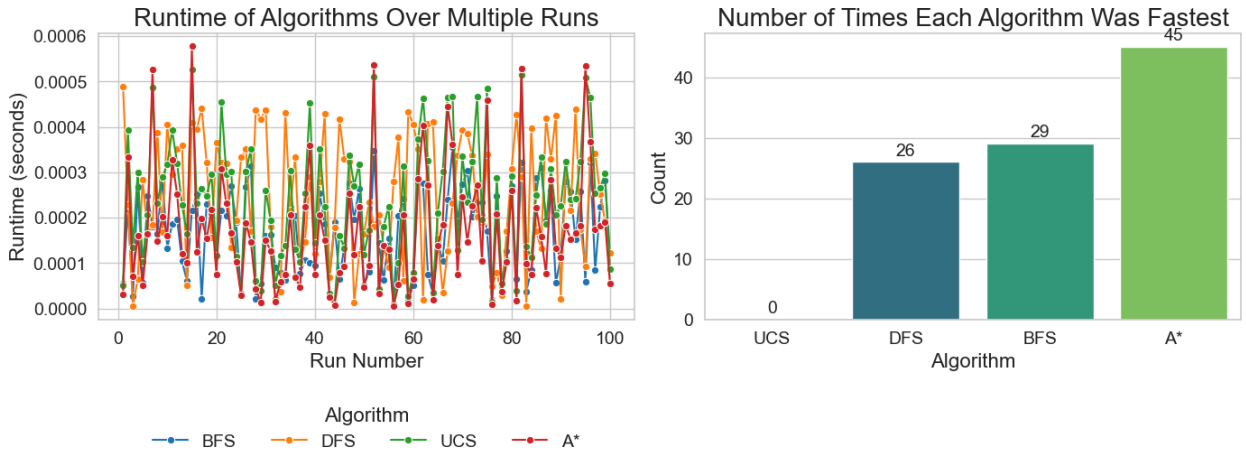
6. Data Analysis and Sources

Data was collected by creating grids of different sizes: 10x10, 25x25, 50x50, 100x100, 250x250, and 500x500. The performance of the 4 search algorithms - BFS, DFS, UCS, and A* was evaluated by running the game 100 times for each grid size.

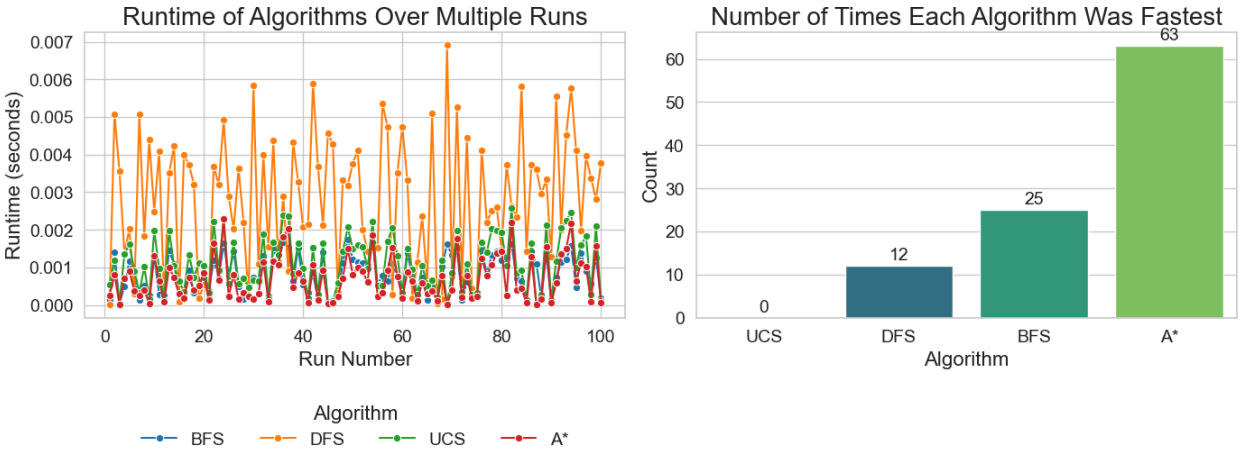
100 Runs on Grid Size 10x10



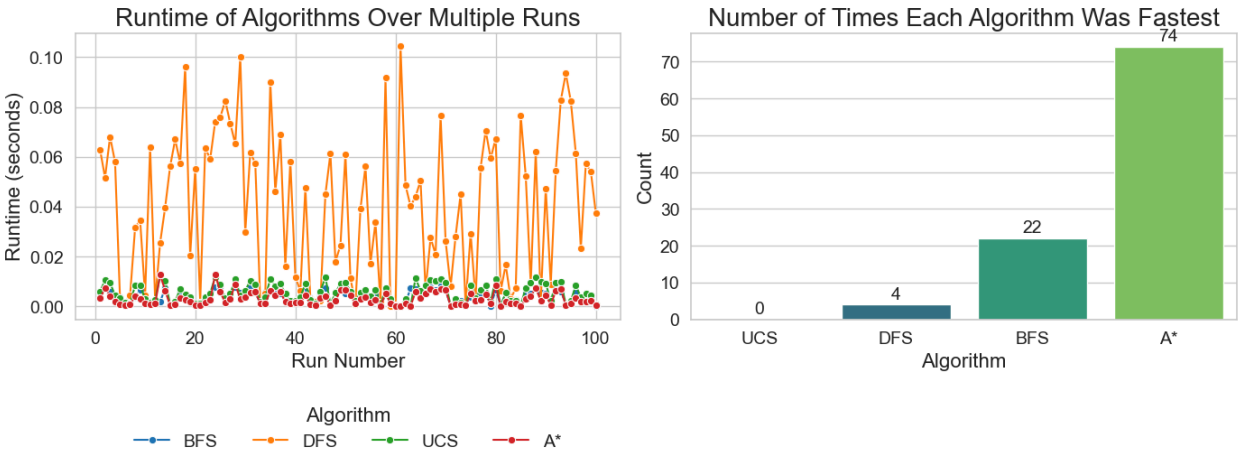
100 Runs on Grid Size 25x25



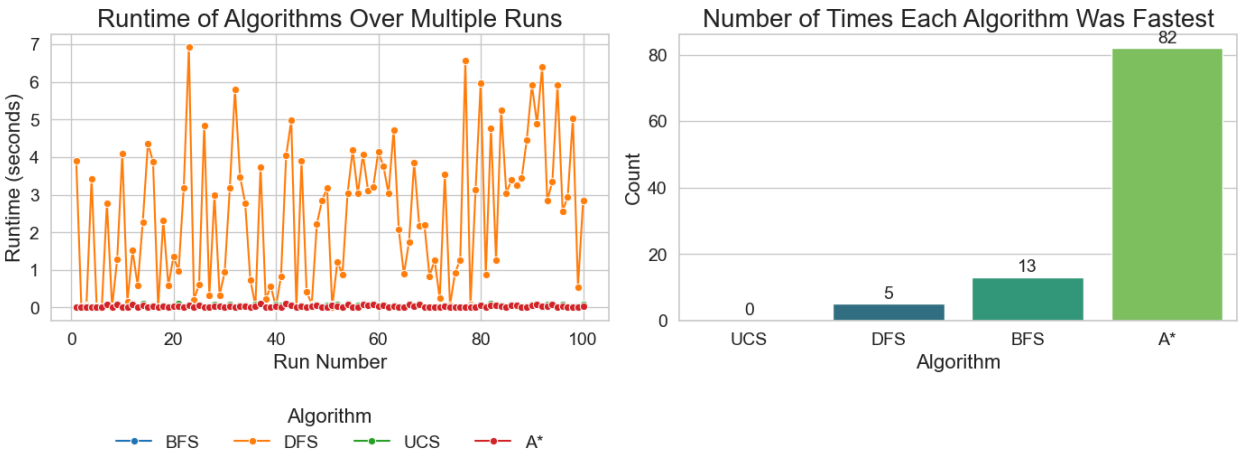
100 Runs on Grid Size 50x50



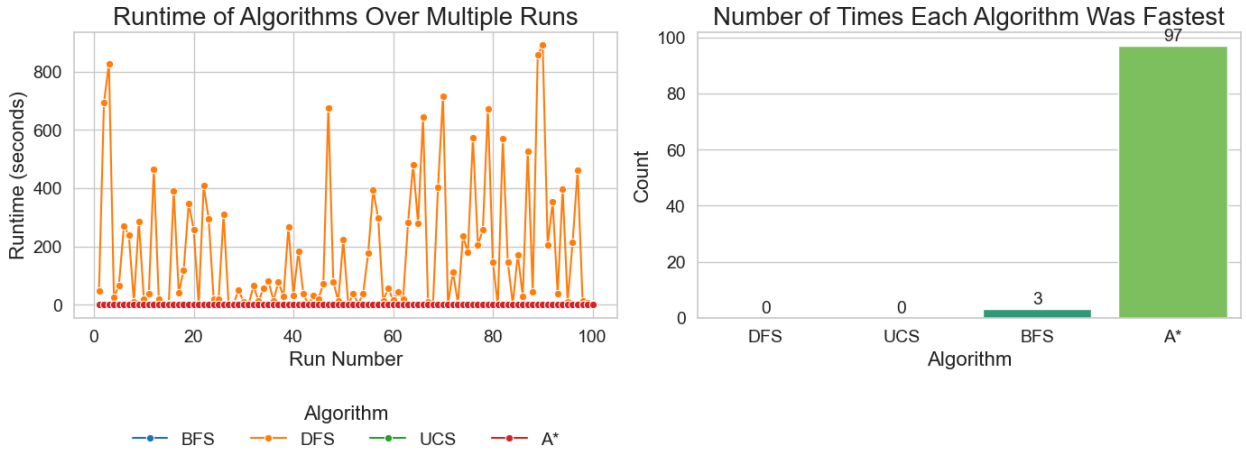
100 Runs on Grid Size 100x100



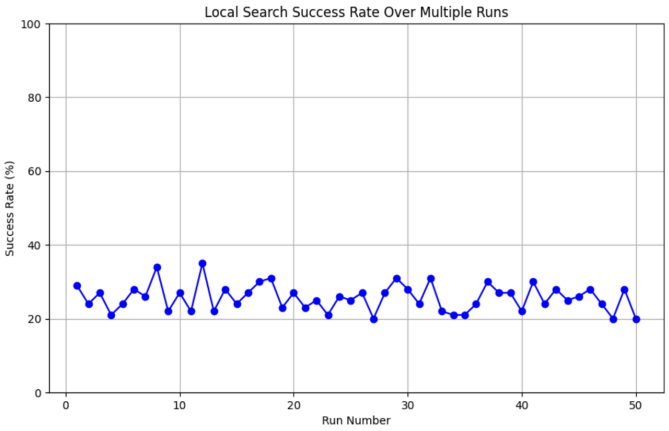
100 Runs on Grid Size 250x250



100 Runs on Grid Size 500x500



Based on the results, we may assume that for smaller grids like 10x10, Depth-First Search (DFS) was the fastest in 44 runs out of 100, then we have A*, which was expected to be the fastest, however, the experiment showed that DFS excels in smaller search spaces due to its ability to find the necessary path without much exploration and it struggles in larger spaces due to its lack of optimality and exhaustive path exploration. As the grid size increased, the picture changed. A* started outperforming the other algorithms being the fastest in 45 runs for 25x25, 63 runs for 50x50, 74 runs for 100x100, 82 runs for 250x250, and 97 runs for 500x500. Although Breadth-First Search (BFS) performed much worse than A*, it was coming the second one after A*. Uniform Cost Search (UCS) did not outperform in any scenario. Compared to DFS and BFS, which could "luckily" find the goal faster due to their simpler strategies, UCS spent more time on finding the most optimal path, which led to a slower result. Also. UCS's lack of heuristic-driven guidance made it much slower compared to A*. making it never be the fastest algorithm out of the four used.



6.7. The experiments conducted involved running local search with random restarts over 50 test runs, each generating 100 random grid maps, yielding a total of 5,000 trials. The success rate per run was recorded, showing considerable variability, as visualized in the provided plot. The average success rate across all runs was 25.72%, with some runs achieving slightly higher or lower performance due to the complexity and randomness of the generated maps. This relatively low success rate indicates that while local search can occasionally find paths to the goal, its performance is highly sensitive to grid configurations, obstacle placements, and the initial starting positions, emphasizing the need for more robust search strategies in such environments.

7. Evaluations

To assess the effectiveness of the search algorithms and logical reasoning, a series of simulations were conducted. Grids of varying sizes (10x10, 25x25, 50x50, 100x100, 250x250, and 500x500) were used to evaluate each algorithm's performance. The metrics for evaluation included time complexity, space complexity, path optimality, and computational efficiency.

8. Conclusions

Hercules' Journey to His Wifey project successfully demonstrated the potential of AI-driven search and logical reasoning techniques in dynamic, grid-like environments. The use of BFS, DFS, UCS, and A* provided a comprehensive analysis of search strategies, highlighting their respective strengths and weaknesses. Empirical results revealed that while BFS and DFS could solve simple grids, A* outperformed all other strategies in larger, more complex environments. One of the key innovations of the project was the introduction of the Hydra as a dynamic and stochastic element. This addition required the agent to incorporate probabilistic reasoning, transforming the deterministic grid environment into a stochastic one. The performance analysis of the search algorithms provided essential insights. A* was found to be the most effective algorithm for large, constrained environments, while DFS showed strength in smaller grids. The Manhattan distance heuristic, used in the A* algorithm, was proven to be admissible and consistent, ensuring optimal and complete solutions.

9. Directions for future research

This problem is remarkable due to its potential for continuous expansion and the integration of new functionalities. Our exploration has covered search algorithms, comparisons between known and stochastic environments, resource limitations, and interactive map definitions. However, the problem's flexible structure opens up numerous possibilities for further development and research.

Future research could explore how belief states might enable the agent to navigate such uncertainty and reach the goal efficiently. Additionally, the problem can be made more complex by incorporating Constraint Satisfaction Problems (CSPs) and logical inference, further enhancing the agent's decision-making capabilities and adaptability in dynamic, unpredictable environments.

9.1. CSPs in Hercules' world

Hercules' world can be modeled as a Constraint Satisfaction Problem (CSP) with variables, domains, and constraints governing movement, resources, logic, and interactions with obstacles. By formalizing it as a CSP, we can model complex interactions, ensure path feasibility, and introduce logical reasoning. Key CSP formulations include positional rules, goal attainment, Hydra interactions, energy management, and optimization objectives. While the theoretical framework for CSPs is provided, implementation, such as backtracking search, was not included in this project.

A backtracking search, often used in CSPs, systematically explores all possible variable assignments, pruning branches of the search tree whenever a constraint is violated. The naive Backtracking Algorithm (BT) is a foundational approach for CSPs. In this method, variables are instantiated one at a time, and their values are checked against constraints. If a constraint is violated, the algorithm backtracks to the previous variable and tries a different value. This process continues until a solution is found or all possibilities are exhausted [van Beek, 2006]. While BT provides a clear framework, it can be computationally expensive, especially for large problems. However, its structured approach to exploring constraints makes it a useful baseline for more sophisticated CSP-solving techniques.

We start modeling the constraints from one of the most fundamental constraints in the world of Hercules - the restriction on where Hercules can move. Each position on the grid is modeled as a variable $X_{(i,j)}$, where $(i,\ j)$ represents the coordinates of the cell. Each of these cells can have a domain of $\{accessible, inaccessible\}$. The constraints ensure that cells containing walls, lava, or a living Hydra are marked as inaccessible. Hercules can only move to adjacent, accessible cells.

The formal constraint can be written as:

$$X_{(i,j)} = inaccessible \Rightarrow cell\ contains\ wall,\ lava\ or\ Hydra$$

Additionally, movement constraints specify that Hercules can only move to adjacent squares. If Hercules is currently at position $(x,\ y)$, his possible moves are constrained by the equation:

$$|x - x_{new}| + |y - y_{new}| = 1$$

This guarantees that Hercules can only move to the four neighboring cells (left, right, up, or down).

The primary objective for Hercules is to reach the goal, which must be accessible and connected to his starting position. To model this, we introduce variables $P_{(i,j)}$ that denotes whether a given position $(i,\ j)$ is reachable. Each position's reachability depends on its connection to neighboring squares. The constraints ensure that if a neighboring square is "reachable," then this square also becomes reachable, which can be defined as:

$$P_{(i,j)} = reachable \Rightarrow$$

$$(P_{(i+1,j)} = reachable)\ or\ (P_{(i-1,j)} = reachable)\ or\ (P_{(i,j+1)} = reachable)\ or\ (P_{(i,j-1)} = reachable)$$

If there is no path to the goal due to walls, lava, or the Hydra blocking the route, then Hercules will need to face the Hydra. If the path remains blocked, the goal is deemed unreachable, and the problem instance becomes unsolvable.

A significant challenge in Hercules' world is the presence of the Hydra. Unlike walls or lava, the Hydra can be defeated, but it requires strategic effort. When the Hydra is encountered, the agent must attempt to kill it from an adjacent square. The initial number of Hydra heads $H_{heads}(t = 0)$ is three, and with every failed attempt, two new heads grow. The probability of successfully cutting off the correct head on the k-th attempt is: $P_k = \frac{1}{H_{heads}(t)} = \frac{1}{3+k}$

If the agent fails to cut off the right head, the number of Hydra heads increases:

$$H_{heads}(t + 1) = H_{heads}(t) + 1$$

To successfully kill the Hydra, Hercules must have sufficient energy and attempt the action multiple times, as killing the Hydra depends on probabilistic reasoning. The energy constraints are discussed below.

Energy is a critical constraint in Hercules' world. Every action Hercules takes (moving, fighting the Hydra, or climbing mountains) depletes his energy. The available energy $E(t)$ at time t is treated as a variable. Each movement to an adjacent square reduces the agent's energy by 1 unit. If Hercules encounters a mountain, he loses 10 units of energy, as climbing a mountain requires significantly more effort. Each attempt to kill the Hydra also costs 10 units of energy. These constraints are modeled as follows:

- Movement cost: $E(t + 1) = E(t) - 1$
- Mountain cost: $E(t + 1) = E(t) - 10$

If Hercules' energy $E(t)$ reaches zero, he can no longer make moves, and the current problem instance is deemed unsolvable. Additionally, the energy required to kill the Hydra must satisfy: $n \leq \frac{E}{10}$, where n is the number of attempts needed to kill the Hydra, and E is the total energy available.

The final objective is to guide Hercules to the goal while minimizing energy consumption and maximizing the chances of success. The objective function is defined as: $min(\sum_{all\ actions} cost(a))$

Where $cost(a)$ represents the energy cost or risk associated with an action a (e.g., movement, fighting Hydra, climbing a mountain). The constraints of reachability, energy usage, and percepts must all be satisfied simultaneously. Hercules must avoid paths that lead to excessive energy loss, except when absolutely necessary (like fighting the Hydra).

9.2. Incorporation of logical inferences in Hercules' world

The complexity of Hercules' journey increases in a partially known environment, akin to the Wumpus World, where agents infer hidden hazards using sensory input [AIMA, 2021]. Hercules' logical reasoning could be enhanced with a knowledge base containing facts like "Monster is in position X" or "If warmth is felt, lava is nearby." Using rules similar to Wumpus World logic, Hercules could infer unseen obstacles, avoid combat, or switch to an informed search strategy once the goal's location is known. Logical inference, such as forward and backward chaining, would enable Hercules to adjust his strategy in real time, navigating dynamically and avoiding unnecessary confrontations.

Future work could introduce more complex, dynamic environments where obstacles and objectives change, reflecting the unpredictability of Hercules' trials. Reinforcement learning could be applied to enable Hercules to learn optimal strategies from experience, while multi-agent setups with intelligent monsters would introduce adversaries capable of predicting his moves. Advanced AI techniques, such as Minimax and Alpha-Beta Pruning, could enhance

strategic depth, creating a richer, multi-agent system that merges mythology with cutting-edge AI.

10. References

[AIMA, 2021] - Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.

Marsland, T. (1991). *Single-Agent and Game-Tree Search* . https://webdocs.cs.ualberta.ca/~tony/TechnicalReports/TR91-16.pdf

van Beek, P. (2006). Backtracking search algorithms. In F. Rossi, P. van Beek, & T. Walsh (Eds.), *Handbook of constraint programming* (pp. 85–134). Elsevier. https://cs.uwaterloo.ca/~vanbeek/Publications/survey06.pdf

TED-Ed. (2018). The myth of Hercules: 12 labors in 8-bits - Alex Gendler. In *YouTube*. The myth of Hercules: 12 labors in 8-bits - Alex Gendler

Ammar, A., Bennaceur, H., Châari, I. *et al.* Relaxed Dijkstra and A* with linear complexity for robot path planning problems in large-scale grid environments. *Soft Comput* **20**, 4149–4171 (2016). https://doi.org/10.1007/s00500-015-1750-1

Amit Patel. *Heuristics*. (n.d.). https://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html

Chugani, V. (2024, September 5). *Understanding Chebyshev Distance: A Comprehensive Guide*. Datacamp. https://www.datacamp.com/tutorial/chebyshev-distance