Analysis Report: Machine Learning, Search Problems, and Logic Programming

Afolabi Oguntuase

January 12, 2025

Contents

1 Introduction

This report presents the implementation and analysis of three key areas in Artificial Intelligence: Machine Learning, Search Problems, and Logic Programming. Each section focuses on distinct tasks designed to evaluate the understanding and application of these concepts.

2 Part 1: Machine Learning

2.1 Linear Regression

2.1.1 Implementation

A Linear Regression model was implemented from scratch using gradient descent. The diabetes dataset from `sklearn.datasets` was used, with features standardised to ensure effective computation. The Mean Squared Error (MSE) loss function guided optimisation.

2.1.2 Results

The training process demonstrated a steady reduction in loss over iterations. The final performance metrics are summarized below:

- Training MSE: `2884.922802987494`.
- Testing MSE: `2894.7400189884493`.

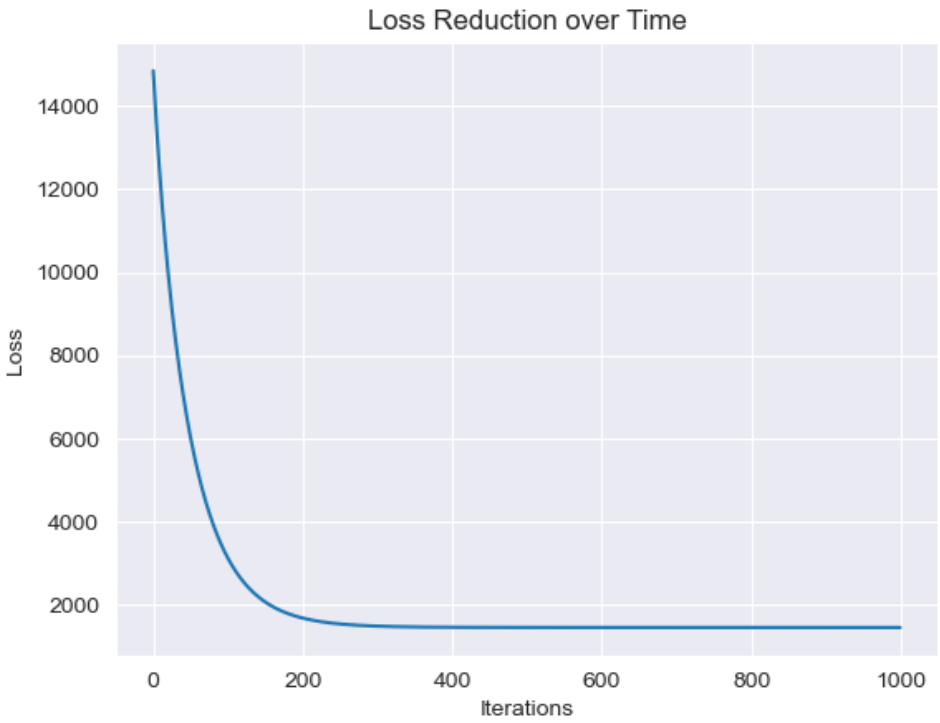

Figure 1: Loss reduction over iterations for Linear Regression.

2.2 Decision Tree Classifier

2.2.1 Implementation

A Decision Tree Classifier was built using the Gini Index for node splitting. The categorical dataset provided was preprocessed to map features to numeric values. The tree structure was visualized graphically.

2.2.2 Results

The model's performance was evaluated using accuracy, precision, recall, and F1-score:

- Accuracy: `0.90`.
- Precision: `1.0`.
- Recall: `0.83`.
- F1-Score: `0.91`.

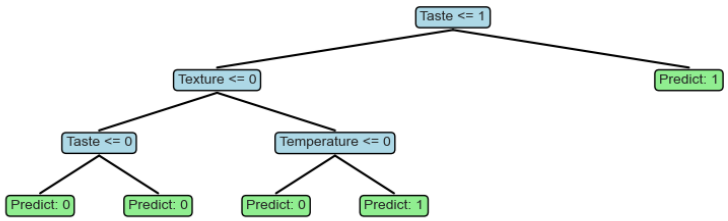

Figure 2: Graphical representation of the Decision Tree.

2.3 Naive Bayes Classifier

2.3.1 Implementation

A multinomial Naive Bayes Classifier was implemented for categorical data. Probabilities were computed based on feature distributions.

2.3.2 Results

Performance metrics for the model include:

- Accuracy: `0.70`.

2.4 Comparison

The performance of all three models was compared using appropriate metrics. Linear Regression minimized MSE effectively, while Decision Tree and Naive Bayes classifiers demonstrated strengths in precision and interpretability.

```
Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.50      0.57         4
           1       0.71      0.83      0.77         6

    accuracy                           0.70        10
   macro avg       0.69      0.67      0.67        10
weighted avg       0.70      0.70      0.69        10
```

Figure 3: Performance metrics for Naive Bayes Classifier.

3 Part 2: Search Problems

3.1 Flight Route Problem

3.1.1 Implementation

A flight route network was simulated, represented as a graph where nodes correspond to cities, and edges represent available flight connections with associated costs (in arbitrary units). The graph structure is as follows:

```
{
    'New York': {'Chicago': 22, 'London': 38, 'Los Angeles': 36, 'Toronto': 21},
    'Chicago': {'New York': 22, 'Denver': 23, 'Dallas': 24, 'Atlanta': 22},
    'Denver': {'Chicago': 23, 'San Francisco': 24, 'Seattle': 25, 'Phoenix': 23},
    'San Francisco': {'Denver': 24, 'Seattle': 22, 'Los Angeles': 21, 'Tokyo': 30},
    'London': {'New York': 38, 'Tokyo': 50},
    'Los Angeles': {'New York': 36, 'San Francisco': 21},
    'Toronto': {'New York': 21},
    'Dallas': {'Chicago': 24},
    'Atlanta': {'Chicago': 22},
    'Seattle': {'Denver': 25, 'San Francisco': 22},
    'Phoenix': {'Denver': 23},
    'Tokyo': {'San Francisco': 30, 'London': 50}
}
```

The BFS, DFS, and A* Search algorithms were implemented to determine the optimal flight route between two cities. A* Search utilized a heuristic function to estimate the remaining cost to the goal. This heuristic assigns a fixed estimated cost to all nodes, effectively prioritizing paths that combine lower cumulative costs with the estimated cost to the destination.

3.1.2 Results

The results for the flight route problem, including runtime and total cost, are summarized below:

Algorithm	Runtime (ms)
BFS	0.032
DFS	0.023
A*	0.029

Table 1: Performance metrics for search algorithms in the Flight Route Problem.

3.1.3 Route Visualizations

Below are visual representations of the routes identified by each algorithm:

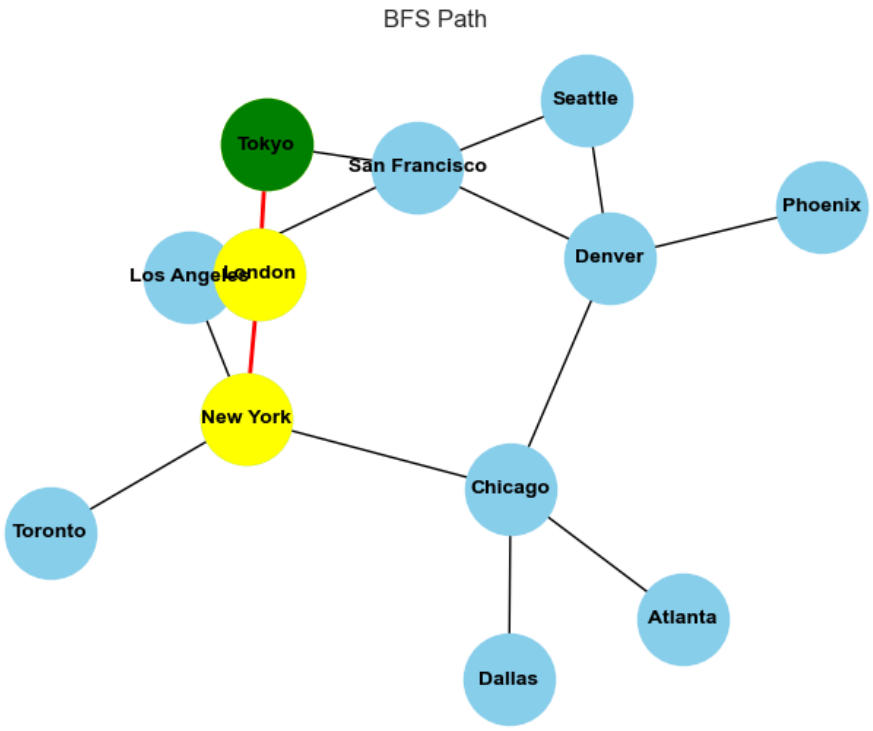

Figure 4: Flight route identified using BFS.

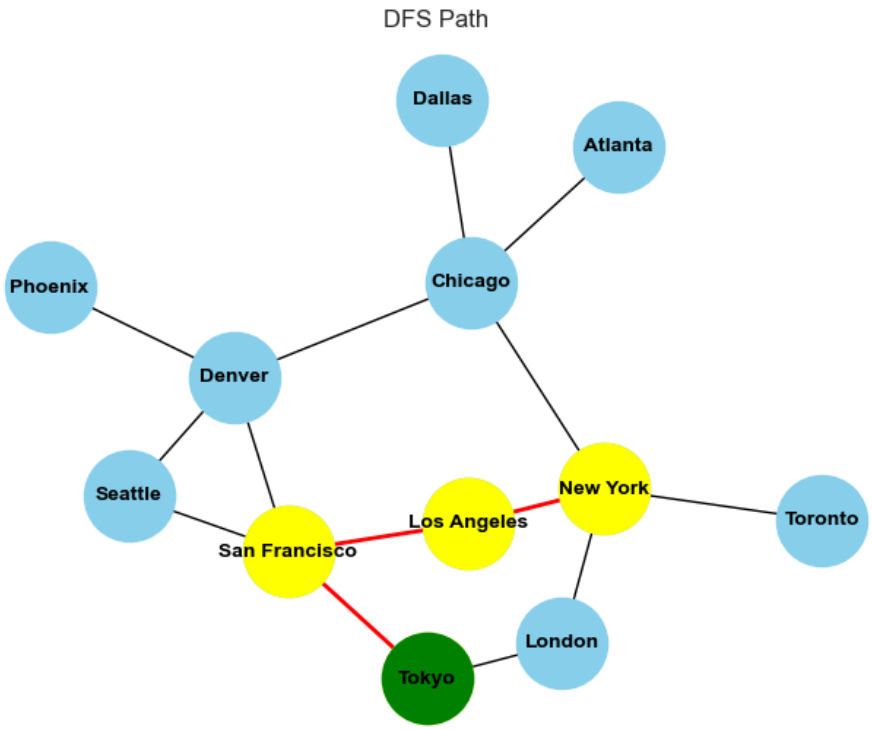

Figure 5: Flight route identified using DFS.

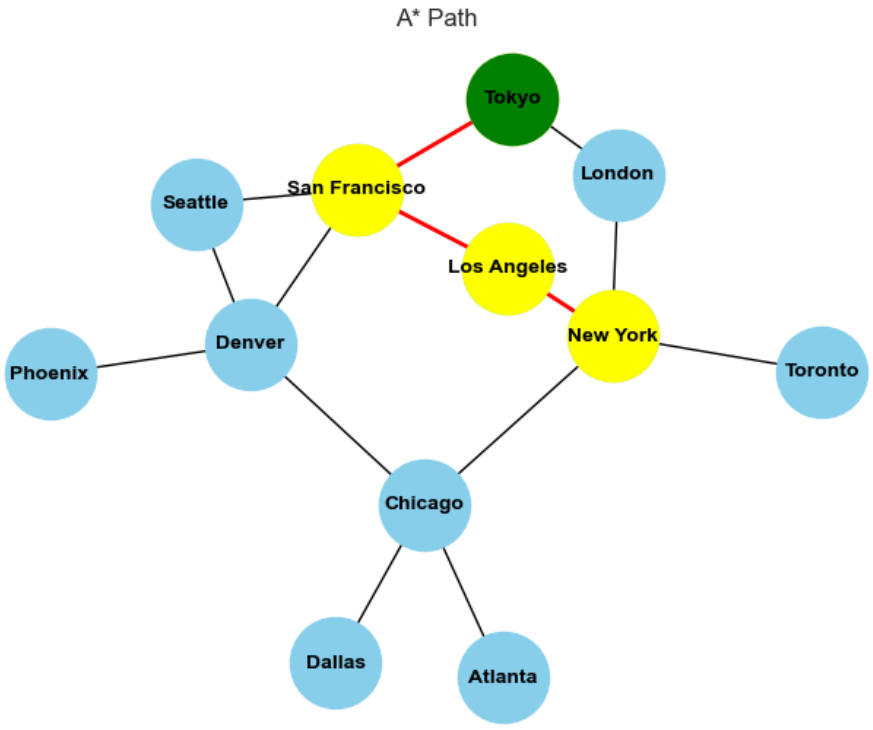

Figure 6: Flight route identified using A* Search.

3.1.4 Analysis

The DFS algorithm outperformed both BFS and A* in terms of runtime, as it was able to explore fewer nodes due to its depth-first nature, leading to faster execution. However, it was less reliable in finding the optimal paths compared to A*, which benefits from the heuristic function guiding the search.

Key observations:

- BFS is exhaustive, ensuring the shortest path is found but requiring exploration of many nodes, leading to higher runtime.
- DFS is faster than BFS and A*, but it can lead to suboptimal solutions due to its depth-first nature.
- A* Search, while guided by heuristics, had the highest runtime because it explores nodes using the heuristic, sometimes expanding unnecessary paths.

3.2 Wumpus World Problem

3.2.1 Implementation

The Wumpus World problem was tackled using BFS, DFS, and A* search algorithms. The goal was to find the optimal path to retrieve gold while avoiding hazards in the environment. The performance of these search algorithms was measured based on their ability to find the path to the gold and their execution time.

3.2.2 Search Algorithm Execution

Below are the results of running the three search algorithms on the Wumpus World problem:

Running A* Search:
A* Path found: [(0, 0), (1, 0), (2, 0), (2, 1), (2, 2)]
A* Execution Time: 0.0010 seconds

Running Depth-First Search (DFS):
DFS Path found: [(0, 0), (0, 1), (0, 2), (0, 3), (1, 3), (1, 2), (1, 1), (1, 0), (2, 0), (2, 1), (2, 2)]
DFS Execution Time: 0.0000 seconds

Running Breadth-First Search (BFS):
BFS Path found: [(0, 0), (1, 0), (2, 0), (2, 1), (2, 2)]
BFS Execution Time: 0.0010 seconds

3.2.3 Results

The execution times and paths for each algorithm are as follows:

3.2.4 Route Visualizations

Below are visual representations of the paths identified by each algorithm in the Wumpus World:

Algorithm	Path	Execution
A*	[(0, 0), (1, 0), (2, 0), (2, 1), (2, 2)]	0
DFS	[(0, 0), (0, 1), (0, 2), (0, 3), (1, 3), (1, 2), (1, 1), (1, 0), (2, 0), (2, 1), (2, 2)]	0
BFS	[(0, 0), (1, 0), (2, 0), (2, 1), (2, 2)]	0

Table 2: Search algorithm execution results for the Wumpus World Problem.

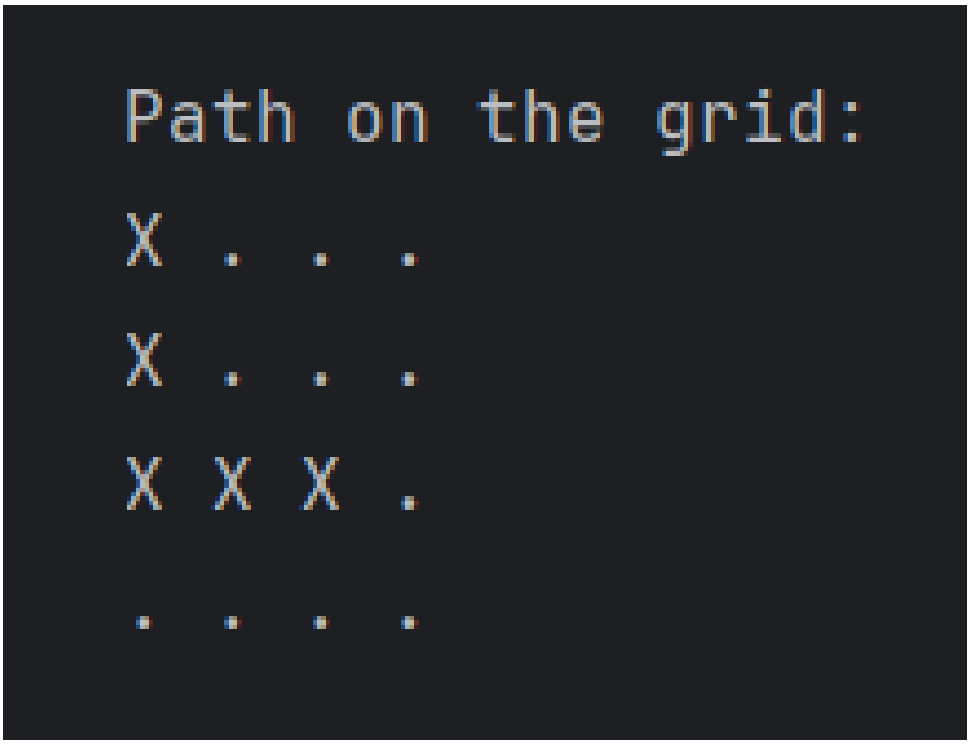

Figure 7: Wumpus World path identified using BFS.

3.2.5 Analysis

In the Wumpus World problem, the DFS algorithm executed the fastest with a time of 0.0000 seconds, but the path it identified was not optimal. A* and BFS took slightly longer, each at 0.0010 seconds, but they produced more reliable and optimal paths. The A* algorithm, leveraging the heuristic function, guided the search towards the goal more efficiently, while BFS explored all possible nodes systematically to find the shortest path.

Key observations:

- BFS, although exhaustive, guaranteed an optimal solution but required more time than DFS.
- DFS, while faster, resulted in a less optimal path due to its depth-first nature.
- A* Search, although taking slightly longer than BFS, provided the most optimal and effective path, making it suitable for more complex problems.

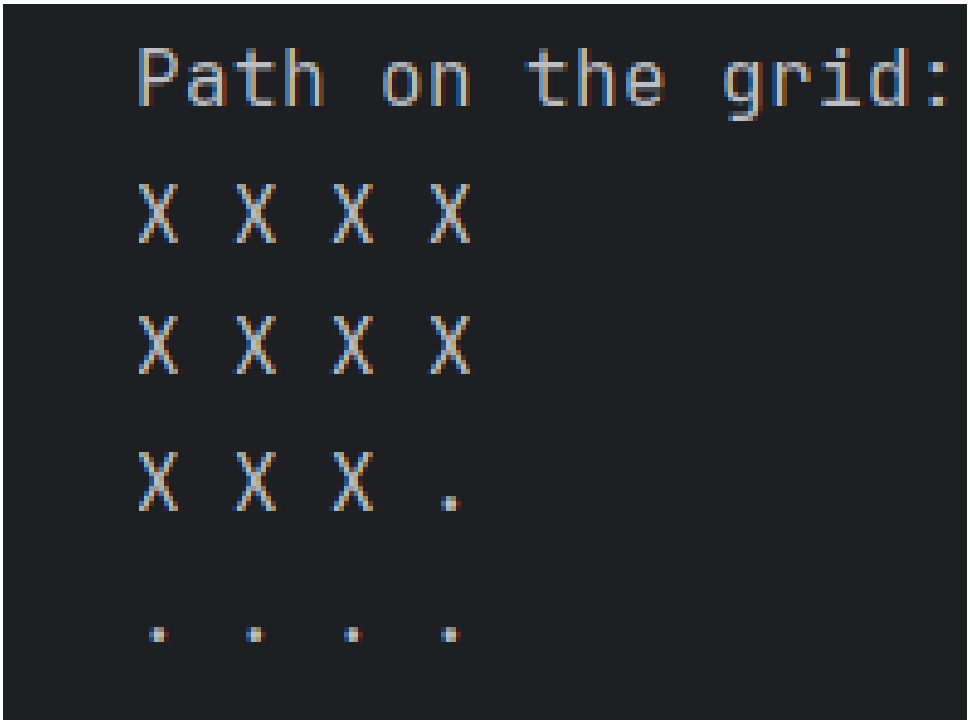

Figure 8: Wumpus World path identified using DFS.

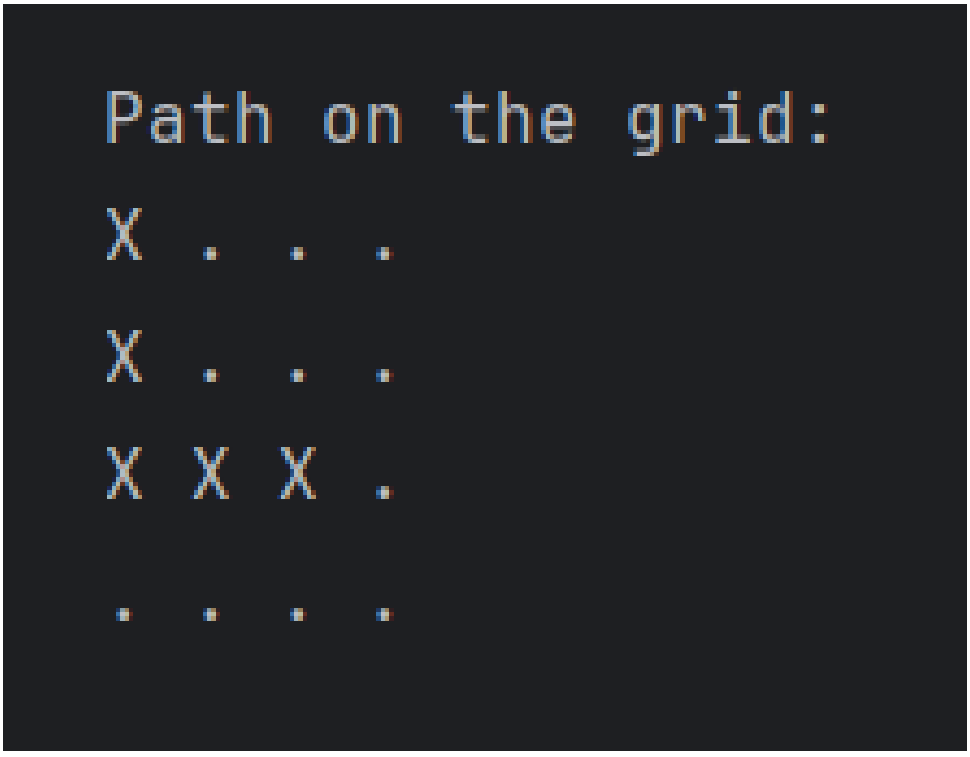

Figure 9: Wumpus World path identified using A* Search.

4 Part 3: Logic Programming

4.1 Wumpus World Problem (Logic)

4.1.1 Implementation

The Wumpus World problem was tackled using logic, pyDatalog was used. The goal was to find the optimal path to retrieve gold while avoiding hazards such as pits and the Wumpus itself. The environment was represented as a grid with the following layout:

Wumpus World Grid:

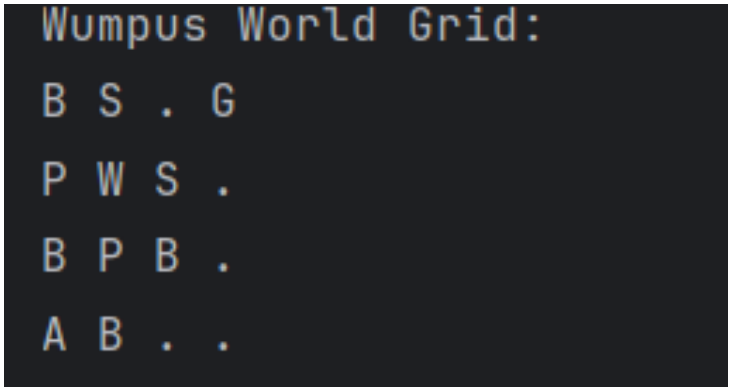

Figure 10: Wumpus World Grid. W represents the Wumpus, G represents the Gold, P represents a Pit, A represents Agent, S represents Stench and B represents Breeze

4.1.2 Analysis

Logical Deduction:
The Wumpus problem, as implemented in the code, demonstrates how propositional and first-order logic can be applied to derive information about the environment:

- The agent determines safe moves by checking its knowledge base, which is dynamically updated based on sensory inputs such as *breeze* (indicating nearby pits) and *stench* (nearby Wumpus).
- The agent marks adjacent cells as safe when no percepts (*stench* or *breeze*) are detected in the surrounding cells, and this decision-making process is reflected in the agent's knowledge base.
- The knowledge base is updated in real-time based on the agent's position and its perceptual input, helping it avoid hazards while navigating the grid.

4.1.3 Complexity

The complexity of the agent's reasoning and decision-making process can be influenced by several factors:

- **Grid Size:** The current implementation uses a fixed 4x4 grid, but if the grid were larger, the number of cells that need to be evaluated would increase. This would lead to a higher computational cost as the agent would need to check more cells for safety and update its knowledge base accordingly.
- **Number of Hazards:** The agent must avoid several hazards (Wumpus and pits). The complexity of the agent's decision-making increases with the number of hazards because it needs to evaluate more percepts and check for potential threats. The code handles a fixed number of hazards (one Wumpus and two pits), but adding more hazards would increase the reasoning overhead.
- **Sensory Inputs:** The agent uses percepts like *stench* (for Wumpus proximity) and *breeze* (for pits) to update its knowledge base. The complexity of handling these inputs increases with the number of percepts and the number of hazards in the grid. The agent needs to check adjacent cells and update its safety status based on these inputs.
- **Dynamic Knowledge Base Updates:** As the agent moves, it dynamically updates its knowledge base, marking safe cells and tracking visited cells. This continuous updating of the knowledge base is a key aspect of the agent's reasoning process, and increases in complexity as the agent explores more cells.
- **Exploration Strategy:** The agent uses a simple exploration strategy, moving to the first available safe adjacent cell. While this is computationally efficient, a more complex strategy could improve the exploration process by prioritizing areas with higher chances of gold or avoiding areas with a high density of hazards.

4.1.4 Key Observations

1. **Logical Reasoning in Partially Observable Environments:** The agent uses logical reasoning based on sensory inputs (*stench* and *breeze*) to infer information about its surroundings. This allows the agent to make decisions about which cells are safe without having full visibility of the environment.

2. **Importance of a Dynamic Knowledge Base:** The agent's knowledge base is updated in real-time as the agent receives new percepts. This dynamic update is essential for consistent decision-making, enabling the agent to adapt to new information as it explores the environment.

3. **Efficiency vs. Completeness of Reasoning:** The agent's reasoning is efficient, relying on simple logical rules. However, this approach lacks the completeness that more advanced inference mechanisms (such as Modus Ponens or probabilistic logic) could offer. The current system does not handle ambiguity or uncertainty in percepts.

4. **Potential for Improved Exploration Strategies:** The agent's current strategy is rudimentary (it moves to the first safe move), which could be improved by incorporating more advanced exploration strategies that consider factors such as the likelihood of finding gold or avoiding multiple hazards.

5. **Uncertainty Handling and Probabilistic Logic:** The agent does not handle uncertainty in percepts, as it relies on deterministic rules. Introducing probabilistic reasoning (e.g., Bayesian networks) could help the agent make better decisions in uncertain situations.

6. **Visual Representation Aids Decision-Making:** The visual representation of the grid helps the agent and the user understand the environment, making it easier to track the agent's actions and the locations of hazards. This improves the agent's interpretability and assists in debugging or refining the decision-making process.

4.2 Scheduling Problem (Logic)

4.2.1 Problem Description

The scheduling problem involves assigning classes to available time slots, ensuring that no two classes overlap and that each class is assigned to a specific lecturer. In this case, we need to assign classes to lecturers based on the time availability and class requirements.

Lecturer, Class, and Time Schedule:

Lecturer	Class	Time
Sophia	Appliied AI	11AM-12PM
Matthew	Intro to AI	9AM-10AM

Table 3: Lecturer Schedule.

4.2.2 Implementation

The scheduling problem was tackled using logic-based constraints. The goal was to assign time slots to the classes and ensure that no time overlap occurred for any two classes. Logic programming constraints were applied to model the problem, ensuring that:

1. Each class is assigned to a specific lecturer. 2. Classes assigned to different lecturers do not overlap in time. 3. The lecturer's time slots are consistent with their availability.

The solution was implemented using logic-based inference to assign the classes to the available slots and validate that no time conflicts arose.

4.2.3 Analysis

The scheduling problem was solved using a logic-based approach, where constraints were formulated and solved using a solver. The approach provided an optimal solution with no conflicts between classes. This method is particularly useful when solving problems with a set of constraints and variables, as it allows for the specification of relationships and ensures that all constraints are satisfied.

4.3 Comparison of Search-Based and Logic-Based Solutions for the Wumpus World Problem

4.3.1 Search-Based Solutions

Search algorithms such as BFS, DFS, and A* were applied to the Wumpus World problem to find the optimal path to the goal while avoiding hazards. These algorithms function by exploring the search space systematically (or based on a heuristic function in the case of A*), identifying paths from the agent's starting position to the goal (the gold) while avoiding dangerous areas (Wumpus and pits).

Strengths of Search-Based Solutions:

- Clear step-by-step exploration of possible states.
- BFS guarantees an optimal solution.
- A* uses heuristics to prioritize paths, leading to more efficient searches.

Weaknesses of Search-Based Solutions:

- Can be computationally expensive, particularly when the state space is large.
- May explore unnecessary states, especially in uninformed search methods like DFS.

4.3.2 Logic-Based Solutions

Logic-based solutions, on the other hand, rely on logical rules and constraints to navigate the environment. In the Wumpus World, this might involve encoding the world's rules (such as the presence of pits and the Wumpus) as logical constraints, and then using a solver to infer the best moves based on those constraints.

Strengths of Logic-Based Solutions:

- Ability to explicitly model and reason about the world using logical rules.

- Can incorporate complex rules (e.g., reasoning about safety) directly into the search.
- Often more flexible and extensible for more complex scenarios.

Weaknesses of Logic-Based Solutions:

- Solving the problem may require advanced inference techniques, which can be computationally intensive.
- May struggle with performance in large search spaces or when too many logical rules are involved.

4.3.3 Analysis of the Comparison

Both search-based and logic-based solutions provide viable approaches to solving the Wumpus World problem, each with its strengths and limitations:

- Search-Based Solutions are more straightforward and provide guarantees for finding a solution (e.g., BFS guarantees the shortest path), but they may struggle with efficiency in large search spaces and can be computationally expensive.

- Logic-Based Solutions offer a more abstract and flexible approach by encoding the world as logical constraints. While this method can handle more complex reasoning and offers a more modular solution, it may face performance challenges due to the computational overhead of logical inference.

In practice, the choice between search-based and logic-based methods depends on the specific problem constraints, the complexity of the environment, and the computational resources available.

5 Conclusion

This report analyzed the implementation of machine learning models, search algorithms, and logic programming. Each approach demonstrated unique strengths, contributing valuable insights into AI methodologies.