Lab Assignment 1
Graph Search Agent for Puzzle-8

Khushi Saxena
Computer Science Engineering
202151078@iiitvadodara.ac.in

Makwana Harsh Maheshkumar
Computer Science Engineering
202151082@iiitvadodara.ac.in

Jagruti Piprade
Computer Science Engineering
202151067@iiitvadodara.ac.in

Nitin Gautam
Computer Science Engineering
202151101@iiitvadodara.ac.in

Python notebook code for this assignment can be found here.

***Abstract*—This report explores the design and implementation of a graph search agent for solving the Puzzle-8 problem. We discuss the problem statement, learning objectives, design of the graph search agent, environment functions for Puzzle-8, Iterative Deepening Search, backtracking function, generation of Puzzle-8 instances, and memory and time requirements for solving them using the graph search agent.**

I. Problem Statement

- Write a pseudocode for a graph search agent.
- Represent the agent in the form of a flow chart. Clearly mention all the implementation details with reasons.
- Write a collection of functions imitating the environment for Puzzle-8.
- Describe what is Iterative Deepening Search. Considering the cost associated with every move to be the same (uniform cost), write a function that can backtrack and produce the path taken to reach the goal state from the source/ initial state.
- Generate Puzzle-8 instances with the goal state at depth "d". Prepare a table indicating the memory and time requirements to solve Puzzle-8 instances (depth "d") using your graph search agent.

II. Introduction

In this report, we delve into the design and implementation of a graph search agent tailored for solving the Puzzle-8 problem. Beginning with an overview of the problem statement and learning objectives, we proceed to detail the design and pseudocode of our graph search agent. Subsequently, we elaborate on the environment functions essential for simulating the Puzzle-8 environment, introduce the concept of Iterative Deepening Search, and furnish a function facilitating path backtracking. Finally, we generate Puzzle-8 instances and furnish a comprehensive table delineating the memory and time requisites for their solution using our graph search agent.

III. Design of Graph Search Agent

Here's the expanded paragraph:

In this section, we provide the pseudocode for the graph search agent and accompany it with a flow chart representation. The design of the agent incorporates the utilization of a hash table and a queue for efficient state space search. Each step of the pseudocode is accompanied by implementation details and reasoning, elucidating the rationale behind the agent's operations. This comprehensive approach ensures a thorough understanding of the agent's functionality and facilitates its seamless implementation in solving the Puzzle-8 problem.

Pseudo Code:

```
1. Start

2. The Agent Takes the initial node and
   goal state from the environment.

3. Agents initialize empty Queue (Frontier),
   HashTable (Visited),
       puts the initial node in the queue.

4. while(frontier(queue) not empty){
       i) Agent dequeue node and add the
       corresponding state to the visited table.
           *check if this state == goal state*
           return "path found"

       ii) get possible actions for the
       current state of the environment
```

```
        iii) For every possible action, the
        agent checks if the corresponding state
        is in the queue and has been visited,
        if not then
            agents enqueues all possible
            nodes(given by environment) to
            queue (added to frontier)
    }

5. return failure
```

IV. Environment Functions for Puzzle-8

In this segment, we furnish a collection of functions meticulously crafted to emulate the environment of Puzzle-8. These functions encapsulate essential functionalities such as generating random instances of the puzzle, verifying goal states to ascertain successful completion, and executing valid moves within the puzzle's state space. By meticulously simulating the Puzzle-8 environment through these functions, we facilitate the seamless integration of our graph search agent, thereby enabling comprehensive experimentation and analysis of its performance in solving the Puzzle-8 problem.

V. Iterative Deepening Search

In this section, we elucidate the concept of Iterative Deepening Search (IDS), a variant of depth-limited search designed to overcome the limitations of traditional depth-first search (DFS). IDS iteratively performs depth-limited searches, gradually increasing the depth limit until the goal state is discovered. By systematically expanding the search space in a controlled manner, IDS ensures completeness while mitigating the risk of exponential space complexity associated with DFS. However, IDS may incur redundant node expansions, particularly in scenarios where the branching factor varies significantly across different depths. Despite this drawback, IDS remains a popular choice for solving problems with unknown or varying depth, owing to its simplicity and efficiency in practice. Through a comprehensive examination of IDS, we aim to provide insights into its efficacy and trade-offs, enabling informed decision-making regarding its application in problem-solving contexts.

VI. Backtracking Function

In this section, we introduce a pivotal function designed to facilitate path tracing within the Puzzle-8 problem-solving framework. This function serves as a crucial component in our graph search agent, enabling the backtracking from the goal state to the initial state and thereby delineating the path taken to achieve the desired outcome. By meticulously traversing the solution path, this function empowers analysts and practitioners to gain deeper insights into the problem-solving process and ascertain the efficacy of the employed algorithmic techniques. Through its seamless integration within our framework, this function plays a pivotal role in enhancing the interpretability and comprehensibility of the solution methodology, thereby contributing to the overall effectiveness of the graph search agent in addressing the Puzzle-8 problem.

VII. Generating Puzzle-8 Instances

In this section, we present a method for generating Puzzle-8 instances with the goal state situated at a specified depth "d", as determined by user input. This functionality enables us to systematically assess the performance of our graph search agent across varying depths of the problem space. By manipulating the depth parameter, we can simulate scenarios of varying complexity, providing valuable insights into the scalability and efficiency of our solution approach. Through rigorous experimentation with Puzzle-8 instances of different depths, we aim to gain a comprehensive understanding of the behavior of our graph search agent under diverse conditions, thereby informing optimization strategies and further refining our solution methodology.

VIII. Memory and Time Requirements

In this phase of our study, we embark on a series of experiments aimed at quantifying the memory and time requirements of our graph search agent in solving Puzzle-8 instances across a spectrum of depths. Leveraging systematic experimentation, we meticulously measure and record the memory usage and runtime performance of the agent under varying levels of problem complexity. Subsequently, we tabulate the collected data and subject it to comprehensive analysis, thereby gaining insights into the efficiency and scalability of our solution approach. By scrutinizing the experimental results, we endeavor to identify patterns, trends, and potential areas for optimization, ultimately informing the refinement and enhancement of our graph search agent. Through this empirical evaluation, we aim to provide a rigorous assessment of the agent's performance and efficacy in addressing the Puzzle-8 problem, thereby contributing to the body of knowledge in algorithmic problem-solving methodologies.

[]

Fig. 1. Memory and Time Requirements

Depth	Time limit(sec)	Memory (KB)
1	0.0001	neglegible
2	0.0001	neglegible
3	0.0001	neglegible
30	0.0015	20
53	0.0039	160

IX. Conclusion

In conclusion, our endeavor has culminated in the development of a comprehensive graph search agent tailored for solving the Puzzle-8 problem. Leveraging a meticulous approach to design and implementation, we have showcased the agent's prowess in efficiently navigating the state space and uncovering optimal solutions. By successfully addressing the intricacies of the Puzzle-8 problem, we have laid a solid foundation for further exploration and application of graph search algorithms in diverse problem domains. Moving forward, our focus will be on refining and optimizing the agent's performance while exploring opportunities for extension to other problem domains. Through continued research and development efforts, we remain committed to advancing the field of algorithmic problem-solving and contributing to the collective body of knowledge in computer science and artificial intelligence.