Gabriel Ferreira
Jin Tian
572 - Principle of Artificial Intelligence
February 18, 2024

Lab 1

(100 points)

Part 2 (20 pts)

Breadth First Graph Search (BSGS)

Solving problem for file: ../Part2/S1.txt
Total nodes generated: 97527
Total Time Taken: 7 min 19 sec 632145 microSec.
Path length: 24
Path: UURDDRULLDRRULLURRDLLDRR

Solving problem for file: ../Part2/S2.txt
Total nodes generated: 29053
Total Time Taken: 58 sec 719342 microSec.
Path length: 20
Path: UURRDLDRULLURRDLLDRR

Solving problem for file: ../Part2/S3.txt
Problem is not solvable.

Solving problem for file: ../Part2/S4.txt
Total nodes generated: 181347
Total Time Taken: 12 min 14 sec 373573 microSec.
Path length: 31
Path: UULDDRRUULDLDRRUULDLDRRUULLDDRR

Solving problem for file: ../Part2/S5.txt
Total nodes generated: 54
Total Time Taken: 1530 microSec.
Path length: 6
Path: LURDDR

Iterative Deepening Search (IDS)

Solving problem for file: ../Part2/S1.txt
Total nodes generated: 63983
Total Time Taken: 4 sec 204923 microSec.
Path length: 24
Path: UURDDRULLDRRULLURRDLLDRR

Solving problem for file: ../Part2/S2.txt
Total nodes generated: 32051
Total Time Taken: 851297 microSec.
Path length: 20
Path: UURRDLDRULLURRDLLDRR

Solving problem for file: ../Part2/S3.txt
Problem is not solvable.

Solving problem for file: ../Part2/S4.txt
Total nodes generated: 596020
Total Time Taken: 32 sec 316005 microSec.
Path length: 31
Path: UULDDRRUULDLDRRUULDLDRRUULLDDRR

Solving problem for file: ../Part2/S5.txt
Total nodes generated: 111
Total Time Taken: 919 microSec.
Path length: 6
Path: LURDDR

A* using Misplaced Tile Heuristic (h1)

Solving problem for file: ../Part2/S1.txt
Total nodes generated: 19382
Total Time Taken: 29 sec 643970 microSec.
Path length: 24
Path: UURDDRULLDRRULLURRDLLDRR

Solving problem for file: ../Part2/S2.txt
Total nodes generated: 3030
Total Time Taken: 665392 microSec.
Path length: 20
Path: UURRDLDRULLURRDLLDRR

Solving problem for file: ../Part2/S3.txt
Problem is not solvable.

Solving problem for file: ../Part2/S4.txt
Total nodes generated: Timed out
Total Time Taken: >15 min
Path length: Timed out
Path: Timed out

Solving problem for file: ../Part2/S5.txt
Total nodes generated: 19
Total Time Taken: 347 microSec.
Path length: 6
Path: LURDDR

A* using Manhattan Distance Heuristic (h2)

Solving problem for file: ../Part2/S1.txt
Total nodes generated: 3188
Total Time Taken: 985615 microSec.
Path length: 24
Path: UURDDRULLDRRULLURRDLLDRR

Solving problem for file: ../Part2/S2.txt
Total nodes generated: 320
Total Time Taken: 12536 microSec.
Path length: 20
Path: UURRDLDRULLURRDLLDRR

Solving problem for file: ../Part2/S3.txt
Problem is not solvable.

Solving problem for file: ../Part2/S4.txt
Total nodes generated: 25947
Total Time Taken: 1 min 17 sec 6085 microSec.
Path length: 31
Path: RUULLDDRRULLDRRUULDRULLDDRRULDR

Solving problem for file: ../Part2/S5.txt
Total nodes generated: 20
Total Time Taken: 322 microSec.
Path length: 6
Path: LURDDR

A* using Max Heuristic (h3)

Solving problem for file: ../Part2/S1.txt
Total nodes generated: 3188
Total Time Taken: 876150 microSec.
Path length: 24
Path: UURDDRULLDRRULLURRDLLDRR

Solving problem for file: ../Part2/S2.txt
Total nodes generated: 320
Total Time Taken: 12753 microSec.
Path length: 20
Path: UURRDLDRULLURRDLLDRR

Solving problem for file: ../Part2/S3.txt
Problem is not solvable.

Solving problem for file: ../Part2/S4.txt
Total nodes generated: 25947
Total Time Taken: 1 min 15 sec 341283 microSec.
Path length: 31
Path: RUULLDDRRULLDRRUULDRULLDDRRULDR

Solving problem for file: ../Part2/S5.txt
Total nodes generated: 20
Total Time Taken: 340 microSec.
Path length: 6
Path: LURDDR

Part 3 (40 points)

Depth	BFGS		IDS		h1		h2		h3	
	Avg run time	Avg #nodes Explr	Avg run time	Avg #nodes Explr	Avg run time	Avg #nodes Explr	Avg run time	Avg #nodes Explr	Avg run time	Avg #nodes Explr
8.00	0.01	134.60	0.004	288.45	0.001	26.30	0.001	23.50	0.001	23.50
15.00	2.46	4,265.10	0.15	8,832.35	0.05	458.10	0.01	193.50	0.01	193.35
24.00	565.81	107,117.70	6.19	295,687.10	62.01	19,485.35	0.51	2,268.75	0.54	2,268.75

Figure 1 To reproduce this table, please run the Part_3.py in the terminal. Once the command finishes running, a .csv and Excel file will be created in the output directory.

Conclusion & Drawn

The table above stores the average run time (seconds) and the average number of explored nodes as performance metrics. In this experiment, five different algorithms were developed to solve the 8-Puzzle and were tested using three different problem depths: 8, 15, and 24.

Here is a summary analysis of the performance of each one of them based on the collected data:

- Breath First Graph Search (BFGS): Although in this scenario this algorithm guarantees to find an optimal solution if there's one, it shows a significant increase in both run time and nodes explored as the depth increases. At the last depth, it had a high average in both metrics, which indicates it doesn't scale well with problem complexity.
- Iterative Deepening Search (IDS): Similarly to BFGS, this algorithm also had its metrics critically increased as the depth of the problem increased. During the development phase, I had a hard time with this algorithm as it took a long time to run. After tweaking some of the code, I was able to make it faster, but it still consumes a lot of memory as it revisits nodes, which leads to a very high number of nodes explored as we can see in the table.
- A* Search: A* Search algorithms, using heuristics to guide their search, had the best performance among the tree families of algorithms with any of the tree heuristics in all depths. Let's analyze the different heuristics:
    - Misplaced Tile (h1): Although this algorithm outperformed BFGS and IDS, it underperformed the other two heuristics.
    - Manhattan Distance (h2): A* Search with Manhattan Distance as heuristics had the best performance along with h3 among all algorithms tested in this lab.
    - Max Heuristic (h3): The Max heuristic is related to the Manhattan Distance as it will choose the heuristic that provides the maximum value between Linear and Manhattan Distance. In this case, Manhattan Distance provided a higher value heuristic than the Linear, and therefore, were selected by the Max Heuristic. This explains the close outcome.

Overall, the A* Search algorithm using different heuristics to guide its solution has proven to outperform BFGS and IDS in terms of run time and nodes explored; especially as the problem becomes more complex.

README.md

Lab1: The 8-puzzle

Problem Description

In this programming assignment, you will write a code to solve 8-puzzle problems. The objective of the puzzle is to rearrange a given initial configuration (starting state) of 8 numbers on a 3 x 3 board into a final configuration (goal state) with a minimum number of actions.

Please Note

All logic and instructions provided below assume that Part2.zip and Part3.zip have been extracted and are available in the directory structure as follows:

```
Lab1/
|-- code/
|       |-- (all logic files and notebooks)
|-- Part2/
|       |-- (all files/states for part 2 of the lab)
|-- Part3/
        |-- L8/
        |-- L15/
        |-- L24/
```

Usage

Components in code directory:

- **output/:** A directory that stores the outputs of part 3 of the lab.
- **search_package.py:** Contains the foundational classes and functions for all algorithms used in this lab. Portions of this code were sourced from the AIMA book repository on GitHub (https://github.com/aimacode) and modified to meet the specific requirements of this lab.
- **part_1.py:** Implements the solution for part 1 of the lab. Utilizes classes and functions from search_package.py. To execute, open a console and run `%run part_1.py`. Follow the prompts to input the file path and desired algorithm. The

printed output will display upon completion of the search. Applies to individual files.

- **part_2.py:** Implements the solution for part 2 of the lab, following a similar structure to `part_1.py` but applies to entire folder. Execute with `%run part_2.py` in the console and follow the prompts.
- **part_3.py:** Addresses part 3 of the lab, automating the solution over multiple puzzles and levels. Running `%run part_3.py` in the console will:
  1. Find 60 8-puzzles. 20 from each of 8, 15, and 24 levels, where the level indicates the optimal path length of the state from the goal.
  2. Solve each puzzle and calculate the average run time and average nodes generated for all five algorithms across each level.
  3. Output the results as tables in both .csv and .xlsx formats within the output folder.
- **part_1_and_2.ipynb:** A Jupyter notebook that mirrors the functionality of `part_1.py` and `part_2.py` but requires manual execution of all cells to generate output.
- **part_3.ipynb:** A Jupyter notebook that mirrors the functionality of `part_3.py` but requires manual execution of all cells to generate output.

Acknowledgments

- [GitHub Pages] https://github.com/aimacode