Task 1 Report

Programming and Mathematics for AI

Éamonn Ó Cearnaigh – 180001598
CITY, UNIVERSITY OF LONDON

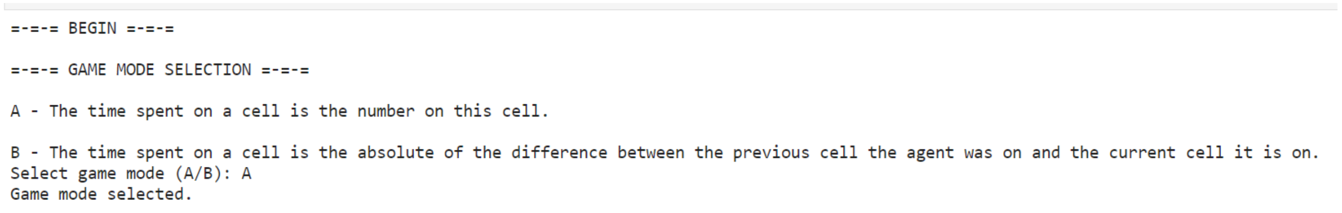
GitHub Repository: https://github.com/EamonnOCearnaigh/StatisticalAnalysis

Game Implementation

The task was implemented using numpy and matplotlib.

It is divided into several functions: a main block (function in the .py file), game_mode_selection, game_parameters, build_grid, baseline_path, and dijkstra_path, and analysis.

The main block first calls the game_mode_selection function, where the user is presented with some instructions, and can select either A or B.



```
=-=-= BEGIN =-=-=

=-=-= GAME MODE SELECTION =-=-=

A - The time spent on a cell is the number on this cell.

B - The time spent on a cell is the absolute of the difference between the previous cell the agent was on and the current cell it is on.
Select game mode (A/B): A
Game mode selected.
```

Validation is provided such that user input is converted to upppercase and must then either match "A" or "B". Therefore, "a" or "b" is also accepted. Relevant variables are returned and passed to the next function.

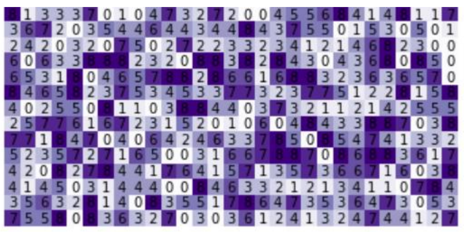
The parameter_selection function is then called, with the user being able to enter values for grid height, width, and max cell integer (n). I set these three to 15, 29, and 9 respectively. I used these values to demonstrate grids without equal width and height. N was arbitrary. Though user input is implemented via the inbuilt input function like game mode selection, commented out in the notebook and python script.

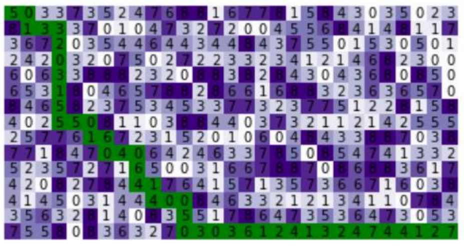
The next function is build_grid, which uses the parameters set by the previous functions to create a matrix with a random seed. An agent is spawned into the top left position, 0,0. It is given a position value of matrix[0][0]. A destination position is set to [width-1, height-1] with value matrix[height-1][width-1] due to the nature of two-dimensional array access in Python. The matrix is visualised using matplotlib. I used a purple colourmap that shades cells based on their n value, and with a green path showing agent movement. I did this by establishing visited cell values to NaN. The inital matrix without a path is displayed.

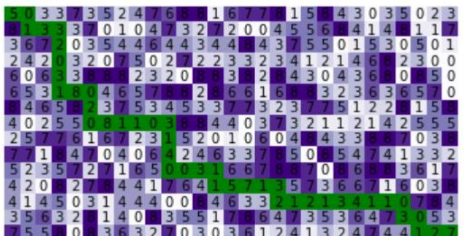
Baseline (heuristic) and dijkstra shortest path algorithms are called (details below) which use the game mode parameter to determine total costs in terms of time (cell value). Matrices are displayed showing the paths taken.

I provided the code to print out agent movement and surrounding cell values for the baseline algorithm though commented this out in the submitted code for clarity. Uncomment for full details of the agent's analysis of the environment.

The image below shows the initial matrix, the matrix with the baseline path, and the matrix with the dijkstra path.







Heuristic Algorithm

A boolean variable controls the algorithm loop by checking whether or not the destination has been reached yet. A total_cost accumulator tracks time spent reaching the destination. While the destination hasn’t been reached, the algorithm checks whether the agent’s position matches the destination position. If so, the loop-break variable is set to true and the loop is ‘continued’ so as to not check out-of-bounds cells in the matrix.

The agent moves diagonally from the top left cell to the bottom right. It does this by either moving right or down, comparing the cell values in each and moving to whichever is lower. When it encounters a matrix-edge (either bottom row or rightmost column) it will only move right or down respectively. Therefore it checks to see whether its x or y co-ordinate matches the destination’s, but not both as this has been accounted for already. Othewise, the cell will check both the cell to the right and below and change position appropriately.

Throughout, the game mode is checked when adding cell values to the total cost. The matrix is then visualised with the agent’s path shown. See image above.

Dijkstra’s Algoirthm

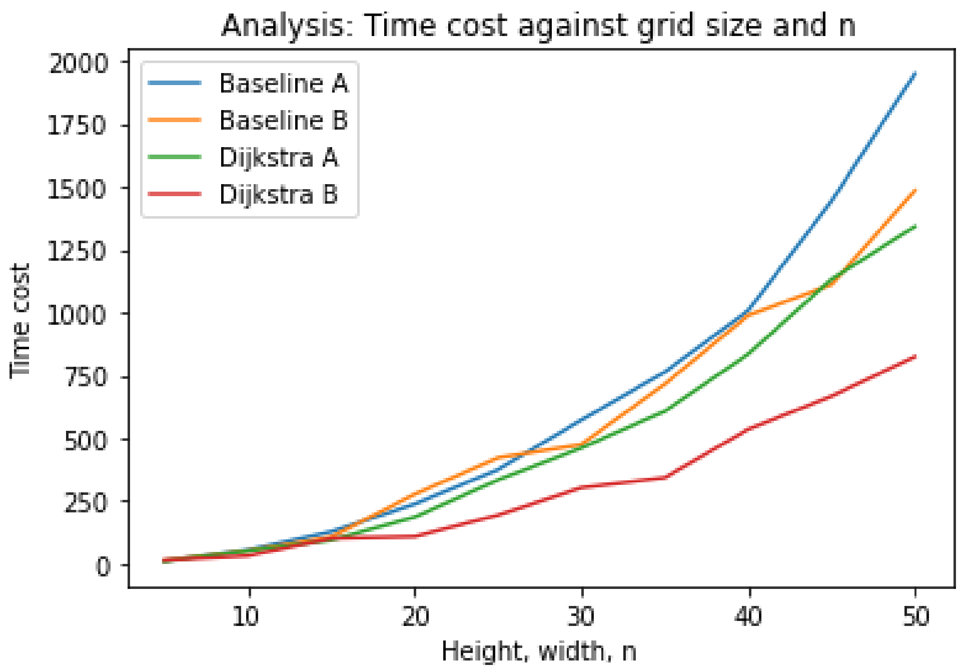
I implemented a shortest-path algorithm based on dijkstra’s algorithm. It uses the same matrix described above to track cell values, a matrix containing the agent’s calculations of the costs for cells as it traverses the matrix, and a boolean matrix tracking which cells have been visited. It examines the cell value of the four cells surrounding an agent. Once it reaches the destination cell, it backtracks, marking the path travelled in the visited_cells matrix, and calculates total cost using the time cost matrix. The matrix and path is visualised (see above). See notebook for implementation details.

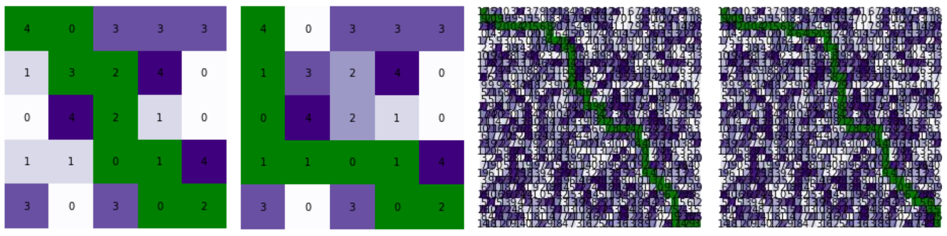
Analysis

For the statistical analysis I automated the running of the established functions with incrementing parameter values. I compared the effectiveness of height, width, n, and game mode on the baseline and dijkstra functions by tracking total cost for grid sizes and n values that were simultaneously increasing by multiples of 5 between 5 and 50. This analysis used grid sizes with height=width=n, with the random generator kept constant. I plotted baseline and dijkstra for both game modes A and B each. The graph below shows the efficiency difference between the approaches, with divergence occuring after width and height 15. An improvement upon this would be to vary the parameters from each other and to analyse them individually in isolation.

I then calculated the mean cost, and standard devation for the four main approaches. All approaches have similar characteristics up to around grid size 15 x 15, n = 15. Divergence then occurs with Dijkstra Mode B performing most efficiently, significantly better than Baseline Mode B. Dijkstra Mode A also performs better than Baseline Mode A for the most part, though note Baseline B’s overtaking of the other paths for height (etc.) = 20 to 25.

	Baseline Mean	Baseline SD	Dijkstra Mean	Dijkstra SD
Game mode A	654.4	611.78	505.3	443.13
Game mode B	564.6	472.66	312.3	265.59





Citations

External sources provided 0% of the actual code for this project, but module labs and Stack Overflow were used as extensive references for Python, Numpy, and Matplotlib syntax.

Matplotlib (2021) Available at: https://matplotlib.org/ (Accessed: November 2021).

Numpy (2021) Available at: https://numpy.org/ (Accessed: November 2021).

Brilliant (2021) Available at: https://brilliant.org/wiki/dijkstras-short-path-finder/ (Accessed: December 2021)