CS520 Project 2 Data and Analysis

Aditya Girish, Rishik Sarkar

November 10 2023

1 Bot Designs and Algorithms

1.1 Initial Setup

This project is similar to the first one in that it involves a bot traversing through a grid maze of cells to save crew members while avoiding aliens. In our project version, we created a 30 x 30 NumPy grid containing 0s and 1s, with 0 representing open cells that the bot, aliens, and crew members can exist/move on and 1 representing closed cells (or walls). We have included the Python project within a single file, *main.py*, which contains all the processes for probabilistic updates, sensing, bot and alien movement functions, version-specific bot simulations, and a thorough testing process for each bot that I shall outline in section 3. In our Python project, we use the following characters to represent each entity:

- 0: Open Cell
- 1: Closed Cell
- 2: Bot
- 3: Alien(s)
- 4: Crew Member(s)

Furthermore, as defined in the project outline, there are 8 possible scenarios involving different combinations of bots, crew members, and aliens. Bots 1 and 2 involve 1 bot, 1 crew member, and 1 alien; Bots 3, 4, and 5 involve 1 bot, 2 crew members, and 1 alien; and Bots 6, 7, and 8 involve 1 bot, 2 crew members, and 2 aliens. Each scenario involves certain similarities and differences in the grid traversal algorithm for the bot, but all of them involve deterministic and probabilistic processes that I shall outline below. I shall go over Bot 1 thoroughly, but for the sake of conciseness, I will only delve into the key changes made to the other bots.

1.2 One Alien, One Crew

As mentioned earlier, Bot 1 and Bot 2 both involve 1 bot, 1 alien, and 1 crew member within a 30 x 30 grid maze.

1.2.1 Bot 1

Bot 1 utilizes a crew sensor and an alien sensor at every single time step to figure out if there is a crew member or alien nearby. On top of that, the probabilities of the crew member and alien being at a particular cell are updated every time the bot and alien move. Bot 1 stores the probabilities of the crew and alien being in a certain cell (i, j) within two probability matrices (represented as a dictionary of open cells + bot's current location) *crew_matrix* and *alien_matrix*. The process for Bot 1's maze traversal is as follows:

1. Initialize a 30 x 30 grid and place the bot, one alien, and one crew member at random locations. The alien is placed outside a (2k + 1) x (2k + 1) square around the bot, and the crew member is placed at any location apart from the bot's current cell. Also, initialize additional metric calculation variables for win_count, loss_count, move (i.e., counter for # of moves per crew member), etc.

2. Check for valid neighbors from the bot's current location (i.e., a list of open cells that the bot can move to at a given time)

3. Determine the best neighbor for the bot to move to (including itself) based on the probabilities in the *crew_matrix* and *alien_matrix*.

    - Bot 1 prioritizes cells with a 0 probability of an alien being in it. Suppose no cells are found with a 0 probability of the alien (reflected in the alien matrix). In that case, the bot prioritizes cells with a non-zero probability of the crew member being in it. This is to prevent the bot from going to cells it has visited before or those that are unlikely to contain a crew member at all (unless necessary, in case of an alien being nearby or if no such cells exist). When the bot finds two neighbors with an equal probability of the crew member being in it, it randomly breaks the tie and picks the next move.

4. The bot moves to the determined best neighbor

    - The number of moves is incremented by 1
    - If the bot enters the crew member's cell at this point, a win point is awarded to it, and a new crew member is placed at another random location on the grid (adhering to the previous constraints). The number of moves is also reset. The win point and the number of moves taken are recorded to be returned as metrics later.
    - If the bot enters the alien's cell or if the number of moves $\geq$ a timeout parameter, a marker is returned depicting that it has been captured or timed out, and a new session begins in the same grid with a newly placed bot, alien, and crew member. Except for the win and loss counts, every other variable is reinitialized (including the alien and crew matrices).

5. After the bot has moved, the alien and crew matrices are updated based on the new beliefs and knowledge:

- Since we know that the new bot location does not contain a crew member (even if it did earlier, it has been teleported away) or an alien, we can set the current cell to have a 0 probability for both crew and alien in their respective matrices (dictionaries).
- We must now normalize the probabilities for the rest of the cells within the two matrices so that the total adds up to 1.

6. Next, the alien moves to a random neighboring cell or stays in place.

    - If the alien enters the bot's cell at this point, a marker is returned depicting that the bot has been captured, and a new session begins in the same grid with a newly placed bot, alien, and crew member. Except for the win and loss counts, every other variable is reinitialized (including the alien and crew matrices).

7. After the alien moves, the alien matrix is updated once again based on the new beliefs and knowledge:

    - When the alien moves, the probability of the alien being in the cell is distributed over all the neighboring cells and the current cell for every cell. The thought process behind this was that there should be an equal likelihood of the alien moving to any of its neighboring cells (or itself) every time the alien move occurs. This allows for a more thorough probabilistic distribution of where the alien might be after it moves.

8. After the bot and alien move, the alien sensor is activated. The bot inputs the k-value of the sensor as a parameter k, and this deterministic value represents the range of the detection square.

    - As mentioned earlier, the detection square is centered around the bot, and it returns a beep (i.e., the function returns *True*) if the alien is detected within the detection range (if alien-x $<$ bot-x + k and alien-x $>$ bot-x - k and alien-y $<$ bot-y + k and alien-y $>$ bot-y - k) and doesn't return a beep if not (i.e., the function returns *False*).
    - The beep value is stored within a variable *alien_detected*.

9. After the alien sensor activates, the crew sensor is activated. The bot inputs the α-value of the sensor as a parameter alpha, which modulates the probability of a beep occurring for a crew member d-steps away. The process taken by the sensor is as follows:

    - The bot maintains a dictionary called *d_lookup_table* that contains every cell the bot has visited as keys and the corresponding *d_dict* dictionaries as values. Each *d_dict* dictionary contains the shortest distance d from the corresponding bot to every other open cell in the grid as values with the open cells as keys.
    - If the bot's current cell has not been visited before, it utilizes a modified Dijkstra's search (using BFS) to calculate the shortest distance d to every open cell in the grid and stores the dictionary as the value with the bot's cell as the key within *d_lookup_table*.
    - If the bot has visited the current cell before (and it is a key within *d_lookup_table*, the *d_dict* is retrieved.

- The lookup table is maintained to avoid running Dijkstra's whenever a bot enters a cell. Although it increases space complexity, the process dramatically reduces the total computation time needed for the sensor to run.
- The *d_dict* is used to find the actual shortest deterministic distance d to the crew member from the bot, and the sensor returns a beep (i.e., *True*) with probability: $e^{(-\alpha \cdot (d-1))}$.
- As the α value increases, the probability of the beep decreases for when d is farther away.
- The beep value is stored within a variable *crew_detected*.

10. After the *alien_detected* and *crew_detected* variables are stored, the *alien_matrix* and *crew_matrix* are updated based on whether a beep was returned for each sensor.

11. The updates occurring within the *alien_matrix* are as follows:

    - If the alien sensor beeps, the bot knows that the alien is within the range of the detection square. Thus, it can be inferred that every cell outside the detection square must have a probability of 0 (since there is only one alien), and the probability of the alien being in any of the cells inside the square must be normalized.
    - If the alien sensor doesn't beep, the bot knows that the alien is not within the range of the detection square. Thus, it can be inferred that every cell inside the detection square must have a probability of 0, and the probabilities of the alien being in any of the cells outside the square must be normalized.

12. The updates occurring within the *crew_matrix* are as follows:

    - If the crew sensor beeps, the bot modifies the probabilities of all the cells within the crew matrix by multiplying the prior probability of the cell containing the crew member by the likelihood of receiving the beep given its deterministic distance d from the bot (i.e., $e^{(-\alpha \cdot (d-1))}$ for the cell's d). The distance d is once again retrieved from the *d_lookup_table* defined earlier (and updated with the crew sensor).
    - If the crew sensor doesn't beep, the bot modifies the probabilities of all the cells within the crew matrix by multiplying the prior probability of the cell containing the crew member by the likelihood of not receiving the beep given its deterministic distance d from the bot (i.e., $1 - e^{(-\alpha \cdot (d-1))}$ for the cell's d).
    - This update is done because the beep being heard is more likely in cells closer to the bot if the crew is there, so the presence of a beep indicates that the crew member might be closer. In the absence of the bot, we do the opposite since the crew member is more likely to be farther away.
    - Finally, the probabilities for all the entries are normalized by dividing them by the sum of all the probability estimates for the total to equal 1.

13. This process continues until the total number of wins + losses equals a maximum iteration threshold. At the end of the simulation loop, the average rescue moves (i.e., sum of move for

every win // total wins), the probability of rescuing the crew (i.e., wins / (wins + losses)), and the average number of rescued crew (i.e., wins) are returned.

This was the most thorough explanation of the processes we took for a single bot. From here on, I shall focus only on the steps taken by the other bots that differ from Bot 1.

1.2.2 Bot 2

Bot 2 takes the exact same approach as Bot 1 for every function except for the one used to calculate the best next move for the bot given the alien and crew matrices.

1.3 One Alien, Two Crew

1.3.1 Bot 3

1.3.2 Bot 4

1.3.3 Bot 5

1.4 Two Aliens, Two Crew

1.4.1 Bot 6

1.4.2 Bot 7

1.4.3 Bot 8

2 Probability Models and Updates

2.1 One Alien, One Crew

2.1.1 Bot 1

Since bot 1 is 1 alien.

2.1.2 Bot 2

2.2 One Alien, Two Crew

2.2.1 Bot 3

2.2.2 Bot 4

2.2.3 Bot 5

2.3 Two Aliens, Two Crew

2.3.1 Bot 6

2.3.2 Bot 7

2.3.3 Bot 8

3 Performance Evaluation

3.1 Average Moves to Save All Crew

- Bot 1 vs. Bot 2
- Bot 3 vs. Bot 4 vs. Bot 5
- Bot 6 vs. Bot 7 vs. Bot 8

3.2 Probability of Avoiding Alien and Saving Crew

- Bot 1 vs. Bot 2
- Bot 3 vs. Bot 4 vs. Bot 5
- Bot 6 vs. Bot 7 vs. Bot 8

3.3 Average Number of Crew Saved

- Bot 1 vs. Bot 2
- Bot 3 vs. Bot 4 vs. Bot 5
- Bot 6 vs. Bot 7 vs. Bot 8

4 The Ideal Bot

5 Bonus

6 Final Thoughts and Relevant Information