CS:520 Partial Sensing using Inference Agent

Project 2 - Partial Sensing

Karan Ashokkumar Pardasani(kp955)
Naishal Chiragbhai Patel(np781)
Himaniben Hareshkumar Patel(hhp46)

October 18, 2021

[1]

karan.pardasani@rutgers.edu
naishal.patel@rutgers.edu
himani.h.patel@rutgers.edu

Contents

1

Glossary

1. Agent 1: The 4-Neighbor Agent from Project 1 that can see all four cardinal neighbors at once.
2. Agent 2: The Blindfolded Agent from Project 1 that bumps into walls.
3. Agent 3: The Example Inference Agent that partially senses the neighbourhood
4. Agent 4: Our Own Inference Agent that partially senses the neighbourhood
5. Agent 5: Agent which builds on Agent 4 and makes use of partial information in terms of probability.
6. N_x: the number of neighbors of cell x.
7. $Sensed_x$: Whether or not cell x has been visited.
8. $Status_x$: Denotes the status of the cell.
    - 1 –> confirmed block
    - 0 –> confirmed unblocked
    - -1 –> unconfirmed
9. C_x: the number of neighbors of x that are sensed to be blocked.
10. B_x: the number of neighbors of x that have been confirmed to be blocked.
11. E_x: the number of neighbors of x that have been confirmed to be empty.
12. H_x: the number of neighbors of x that are still hidden or unconfirmed either way.
13. Agent Grid : Grid updated by the agent drawing on the knowledge gained using its field of view during Repeated Froward A*. Initially, it is assumed that all cells are unblocked in this grid.
14. Known Grid : Grid containing full knowledge of the environment.

15. Final Discovered Grid : Grid formed after Repeated Forward A*, containing the knowledge of all cells exposed to agent's field of view. Also, all unexposed cells are considered blocked.

16. Heuristic :

    - g(n) : shortest path length discovered from the *start cell* to *current cell*.
    - h(n) : heuristic value which estimates the remaining distance from *current cell* to *goal cell*.
    - f(n) : g(n) + h(n), i.e. total path length estimate from *start cell* to *goal cell*

17. Manhattan Distance : d((x1, y1),(x2, y2)) = $|x1 - x2| + |y1 - y2|$

18. Traversed Path : Cells explored by agent while traversing planned path in all Repeated A* cycles.

19. Trajectory Length : Length of traversed path(with backtracking).

2

Introduction

2.1 Problem Statement

Here, we design and implement agents that travel a grid of 101 $\times$ 101 matrix where the probability of each cell being blocked is p. The agent starts from upper left cell (0,0) and the target cell is lower right cell (100, 100) and the neighbourhood of each cell consists of cells in 8 directions - (N, S, E, W, NE, NW, SE, SW). In this report, we have analysed the performance of Agent 1(4-Neighbor Agent from Project 1), Agent 2(Blind Agent from Project 1) and the following agents in the grid world.

1. Agent 3: A blind folded agent which can sense its surroundings and obtain the number of cells which are blocked in its immediate neighbourhood. It considers individual inferences to confirm the status of unexplored cells and traverses the gridworld using this inferred information.

2. Agent 4: The only difference between Agent 3 and Agent 4 is that Agent 4 takes into consideration multiple individual inferences and obtain concrete information about the status of unexplored cells to traverse the gridworld.

3. Agent 5: This agent builds on Agent 4 and takes into consideration the partial/incomplete information about the environment and uses this in terms of probability of a cell being blocked to plan a path to the goal state. Essentially, the probability of a cell being blocked is a kind of guidance i.e. heuristic which helps the agent plan using incomplete information which we often find in the real world.

In all the three agents, the inference is done using the knowledge gained after entering a cell. All the agents are able to sense the number of cells that are blocked in the neighbourhood of the current cell and this information is used in different ways to gain more knowledge about the grid. Finally, we evaluate the performance of these 5 agents on different metrics.

2.2 Grid Environment

The grid environment is defined by constraint of knowledge. The agent gains knowledge of the grid as it traverses through the grid via inference or by en-

countering blocks.

2.2.1 Cell States

- Blocked cell : Cells agent can not pass through.
- Unblocked cell : Cells agent can occupy and thus move further.

2.2.2 Parameters

The following parameters regulates the formation of the Grid World.

- Grid Dimension : It determines the size of the Grid World.
- Blocked cell density(p) : Its the probability that a given cell will be blocked. For all the analyses, the value of p ranges from $(0, 0.33)$.
- Start cell : Initial starting point in the Grid World for the agent. The start cell of the agent is (0,0)
- Goal cell : End Target of the agent. The target cell of the agent is (100,100)

3

Agent 3

Agent 3 is a blind folded agent which senses its surroundings in 8 compass directions and obtains the number of cells which are blocked in its immediate neighbourhood but their exact location. It makes use of individual inferences using the following Inference Rules as a model to add findings to the knowledge base and confirm the status of unexplored cells in *agent_grid* and uses this information to traverse the grid to reach the goal.

1. $C_x = B_x$: all remaining hidden neighbors of x are empty
2. N_x - $C_x = E_x$: all remaining hidden neighbors of x are blocked
3. H_x= 0: nothing remains to be inferred about cell x

3.1 Knowledge Representation

1. **class Agent3**:
   - **Agent Grid**: 2-D array that represents current knowledge of the agent about the grid.
   - **Grid**: 2-D array that represents full knowledge of the grid. This grid is used when the agent wants to know whether the cell it enters is blocked or not. Also, this grid is used to sense the partial information that agent 3 uses to draw inference.

   The blocked cells are represented as **0** and the unblocked cells are represented as **1** in the above two grids.
2. **class Inference_Agent3**
   - **Agent Grid Object**: Object representing the agent whose current knowledge is used to draw inference
   - **q**: Queue that stores the cells that the inference agent processes to propagate the updates in the agent's knowledge
3. **class GridWorld**:
   - N_x: Number of neighbours in the neighbourhood of cell x.

- C_x: Total number of blocked neighbours
- B_x: Total number of confirmed blocks in the neighbourhood of x.
- E_x: Total number of confirmed empty cells in the neighbourhood of x.
- H_x: Total number of hidden cells in the neighbourhood of x.
- sensed_x: Set of cells that have been sensed by the agent. The agent knows the value of C_x for all the cells present in this set.
- status_x: Dictionary that stores the status of all the explored and unexplored cells which is initially empty. The key is the tuple representing the cell and the value is from the set {-1,0,1}, where:
  - 1 –> cell is blocked
  - 0 –> cell is unblocked
  - -1 –> cell is hidden

  If a cell is present in this dictionary, it means that either the cell is sensed or the cell has been updated by the agent.

4. **class MyPriorityQueue**

- **current_heuristic**: Variable used to specify the heuristic that should be used to calculate priority. **m** denotes Manhattan distance, **p** represents Manhattan distance plus Probability of a cell being blocked(will use this for Agent 5)
- **_data**: Object of class Sorted Set which is used to store cells of the priority queue.
- **g**: Dictionary used to store the g(x) for each cell x. It represents the distance of the cell x from source.
- **h**: Dictionary used to store the h(x) for each cell x. It represents the estimation of the distance of the cell x from target.

3.2 Knowledge Base

The knowledge base of the agent 3 consist of three equations:

1. $B_x = C_x$ If this equation is satisfied, it means the agent has the knowledge of all the blocked cells in the neighborhood. And all the unconfirmed cells in the neighborhood are not blocked.

2. $N_x - C_x = E_x$
   $N_x - C_x$ is the number of unblocked cells in the neighborhood. If this equation is satisfied, it means that the agent has the knowledge of all the unblocked cells in the neighborhood and all the hidden cells in the neighborhood are unblocked.

3. $H_x = 0$ means nothing remains to be inferred about cell x and its neighbors as all neighboring cells have been determined to be blocked or unblocked.

3.3 Implementation

Intuitive steps for Agent 3 implementation:

1. The agent starts from cell (0,0). According to the free space assumption, all the cells of the *agent_grid* are unblocked initially.

2. A path is planned using the A-star algorithm utilising the knowledge obtained by inference and encountering blocks.

3. The agent traverses the cells along it's planned path. At every step, the agent updates it's current knowledge and partially senses information from the current cell. After that, the agent uses this information to infer whether the unexplored cells are blocked or not.

Pseudocode for Agent 3 inference:

1. When the agent starts to explore the cell, at every step, the algorithm calls *infer* method to start inference from the current cell.

2. When the agent first enters the cell, there can be three cases for the current cell: 1)visited, 2)inferred, 3)hidden.

   (a) If the current cell is visited - then it will not be blocked and no extra knowledge will be inferred.

   (b) If the current cell x is inferred - then the agent will find C_x from the known grid. Now, if the knowledge base conditions are satisfied on x, the hidden neighbours of x will be marked blocked(1) or unblocked(0) and the changes are propagated to the rest of the grid. *The algorithm of propagation will be explained after few steps*

   (c) If the current cell x is hidden, then the agent will find C_x from the known grid and checks whether the base conditions is satisfied or not. Based on the satisfiability of knowledge base conditions, the algorithm will propagate the information to the rest of the grid.

3. After the agent propagates the information to the rest of the grid, it checks whether the planned path for next 4 cells is blocked or not. If so, then the agent stops traversing the current planned path and re-plans the path using A-star algorithm from the current cell.

Propagating the Information in the grid

1. If the agent enters a cell which is already visited, it does not propagate any information since the agent is not receiving any additional information.

2. If the current cell is inferred, then it is not required to mark that cell or update it's neighbourhood, since the agent already know whether the cell is blocked or not. Also the inference engine updates the information of the current cell for it's neighbours. The agent is only required to calculate the value for C_x and check for knowledge base conditions in x.

3. If the current cell is not inferred, then the agent marks the current cell and the inference engine will update it's neighbours to account for the current cell to be blocked or unblocked. If the current cell is blocked, the

inference agent will decrease the value H_x and increase the value for B_x. If the current cell is unblocked, the inference agent will decrease the value H_x and increase the value for E_x. After the agent updates H_x, B_x and E_x, the agent checks whether the knowledge base conditions are satisfied for the current cell x.

4. In the steps 2, 3 If the knowledge base conditions are not satisfied, then the neighbours for which E_x, B_x and H_x is updated and are also visited are added to the queue.

5. In the steps 2, 3 If the knowledge base conditions are satisfied, then the function *check_kb()* will return 1 if the hidden neighbours are blocked and *check_kb()* will return 0 if the hidden neighbours are unblocked. Then the agent will mark the hidden neighbours blocked or unblocked according to the value of *check_kb*. Also, since the hidden neighbours are marked, their neighbour's E_x, B_x and H_x needs to be updated. The algorithm will push all the visited cells whose E_x, B_x and H_x is updated in the queue.

6. It is important to push only updated and visited cells in the queue because if the cell is not visited the inference engine will not be able to check knowledge base equations without the value of C_x. Also, if the value of E_x,H_x and B_x is not updated then there is no variable whose value is changed in the knowledge base equations. Hence, if the H_x,E_x and B_x is not updated then the agent does not push the cell in the queue.

7. After we push the cells in steps 4,5, we know that these cells are updated and visited. So, the agent needs to check the knowledge base conditions of the cells. If the *KB* condition is not satisfied or the number of hidden cell is 0 in the neighbourhood of popped cell, the agent has finished processing this cell.

   If the KB condition is satisfied for the popped cell y, then the agent marks all the hidden neighbours of y as blocked or unblocked. Also, the agent updates the values of E_x,B_x and H_x accordingly. Once these updates are completed, the agent pushes these cells to the queue.

8. The agent also needs to keep track of the cells present in the queue. This is because if a cell is already present in the queue and it again gets updated in another neighbourhood, it is not required to add the cell to the queue again. To achieve this, the code contains a set data structure called **present**. When we add a cell to the queue, we add it to the **present** set and when we pop a cell from the queue, we remove the cell from the **present** set.

9. The agent keeps on popping cells from the queue until the queue gets empty.

3.4 Comparing Agent 3 with Agent 1 and 2

3.4.1 Number of Bumps

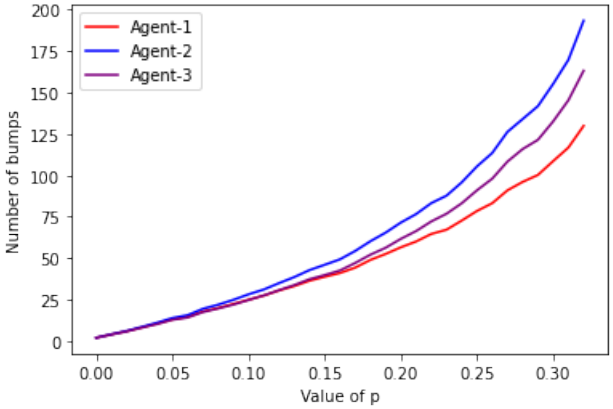


Figure 3.1: Number of Bumps vs Density

Agent 1 has concrete information about its neighbourhood while Agent 3 has partial sensing and needs to infer the information. Thus Agent 1 has more information about its environment and thus does a better job at planning and encounters less blocks as a result. Also, Agent 2 is blindfolded and thus has the least amount of information available and therefore performs the worst in terms of blocks encountered. We will see a more detailed study of the results and comparison of all 4 agents in chapter 4.

Note: *Through this, we can also infer that the amount of information available to Agent 1 > Agent 3 > Agent 2.* This is a very important conclusion and we will use this as one of the basis of many our deductions.

3.4.2 Number of Cells Processed

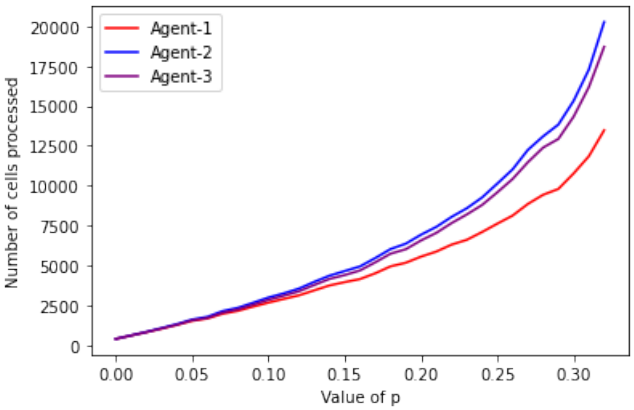


Figure 3.2: Number of Cells Processed vs Density

As we figured in previous paragraph, Agent 2 has the least information about the environment and thus needs to roam around more to gather quality information to reach the goal.Therefore the number of cells processed for Agent 2 is the highest following Agent 3 which has more information than Agent 2 and finally Agent 1 which has most definitive information.

3.4.3 Number of Discovered Cells

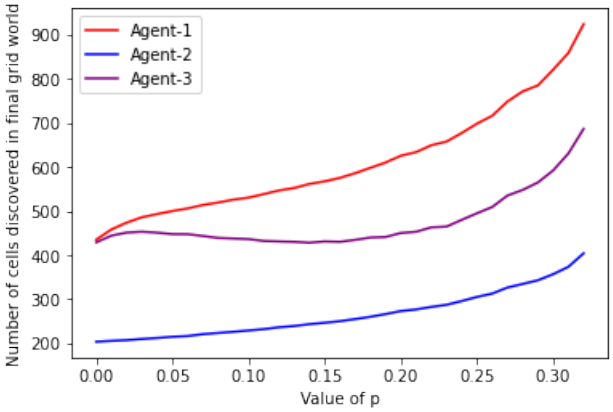


Figure 3.3: Number of Discovered Cells vs Density

As expected, the number of cells inferred by agent 3 in the final discovered gridworld is in between agent 1 and agent 2 as we inferred above that the amount of information available to Agent 1 > Agent 3 > Agent 2 .The ability of Agent 4 to combines individual inferences to make new inferences outperforms not only agent 3 inference but also the definitive knowledge of agent1 as Agent 4 can infer information about far away cells and thus is capable of discovering more cells.

3.4.4 Trajectory Length

Surprisingly, Trajectory length is the only metric on which all three agents are performing the same with very little variation despite them having different level of sensing and inference capabilities. This might be because of the fact that all the agents are not able to tap on all parts of vital information and are missing out on some information. For example, Agent 1 has concrete information about 4 neighbours and Agent 3 has partial information about 8 neighbours which is not enough to reduce backtracking and thus trajectory lengths are similar. It is not clear from the above graph, but analysing the data for this graph shows us that Agent 1 performs slightly better than Agent 3 followed by agent 2 which is quite expected.

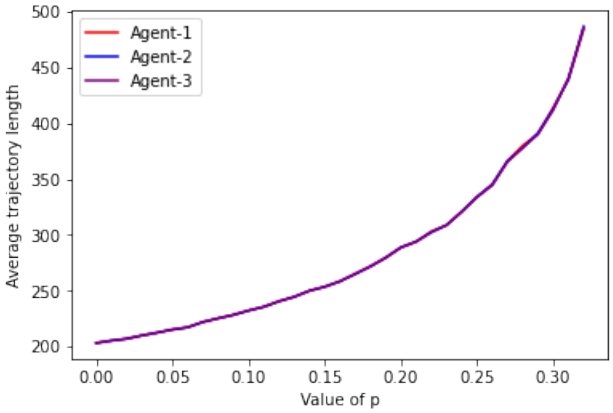


Figure 3.4: Trajectory Length vs Density

3.4.5 Path Length

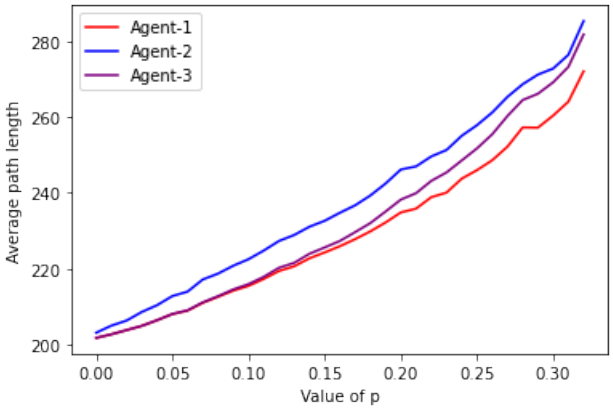
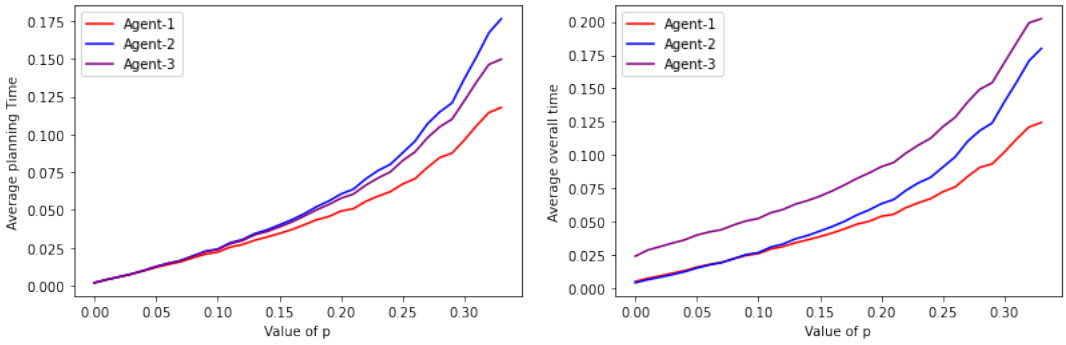


Figure 3.5: Path Length vs Density

The shortest path travelled by the agent on Final Discovered Grid is Path Length. We observe that the Path Length for Agent 3 is less than Agent 2 and more than Agent 1. We also know from *Figure 3.3: Number of Discovered Cells vs Density* that Agent 1 has the highest discovered cells in Final Discovered gridworld following Agent 3 and Agent 2. Thus Agent 1 has access to more information about the Known Grid. Thus less of the available paths are considered blocked in the final discovered gridworld and thus the agent is able to find shorter path to the goal beating agent 3 in this case. Thus, *Quality information available to Agent4> Agent 1> Agent 3> Agent 2*

3.4.6 Planning Time and Overall Time

From *Figure 3.2: Number of Cells Processed vs Density* we can see that Agent 1 processes the least number of cells following Agent 3 and Agent 2. This implies that Agent 1 has to process less cells during the planning phase which reduces



(a) (b)

Figure 3.6: (a) Planning Time vs Density (b) Overall Time vs Density

its planning time and the same for other agents. Agent 3 still ranks second on this metric.

Next we observe that Agent 3 ranks last on overall time taken. This is because Agent 3 needs to make inferences while Agent 1 and 2 has to make no inferences and thus their overall time is divided into planning and execution while the time taken for Agent 3 includes inference in addition to planning and execution.

4

Agent 4

Agent 4 is also a blind folded agent which senses its surroundings in 8 compass directions and captures same information as that of agent 3. Agent 4 is an advanced version of agent 3 that infers more than what the Agent 3 does. The inference engine in Agent3 just looks at individual equations containing neighbourhood of x but in Agent 4, the inference engine combines different equations and draws inferences about unexplored cells. Two different techniques are used to combine equations present in knowledge base:

- Simplifying using Subsequence
- Simplifying using Set Difference

4.1 Approach for the problem

By analysing the knowledge base conditions of Agent 3, it can be seen that the relations between H_x, E_x, B_x and C_x, can also be captured using the following equation:
Consider the neighbourhood of cell x which contains 8 neighbours namely, A, B, C, D, E, F, G, H and the book-keeping values for cell x is C_x, B_x, E_x and H_x.

$$A + B + C + D + E + F + G + H = C_x \tag{4.1}$$

If neighbours A, C, F are blocked and G is unblocked, then we can see that E_x will be 1 and B_x will be 3. And the above equation will become:

$$B + D + E + G + H = C_x - 3 = C_x - B_x \tag{4.2}$$

Everything that agent 3 stores for inferences can be captured in this equation. We can see that the *len(LHS)* is equal to H_x. $B_x = C_x$ - RHS and $E_x = N_x$ - H_x - B_x (Number of confirmed cells = E_x + B_x = N_x - H_x). For e.g., consider the two equations:

$$A + B + C + D = 3 \tag{4.3}$$

$$B + C = 1 \tag{4.4}$$

We can put the value of second equation into the first equation:

$$A + D = 2 \tag{4.5}$$

Hence, instead we can replace equation 4.3 with equation 4.5 in knowledge base.

Simplifying using Subsequence

When we tried to write some equations for small random grids, we observed that sometimes we can put the value of a smaller equation to reduce the length of a longer equation. This is the main idea of the technique of **Simplifying using Subsequence**

Simplifying using Set Difference

Also, we observed that when the number of variables with positive signs in an equation is equal to the RHS, then all the variables with positive sign will be 1 and all the variables with negative sign will be 0. Consider the following equation:

$$A + B + C - D - E = 3 \tag{4.6}$$

In the above equation, if atleast one of the variable with positive sign will be 0, then the equation cannot be solved. Hence it is required for each of the variables with positive sign t be 1. And also, we can see that the variables with negative sign will be 0.

4.2 Knowledge Representation

To draw inference, agent 4 forms equation at cell x, that consists of hidden cells and the RHS contains the value $C_x - B_x$, i.e. the number of neighbouring blocks that have not been confirmed. All the equations formed is stored in the knowledge base. The data structure used to store equation is a list that contains two elements: A set and an integer. The set consists of cells (as tuples) and the integer is the value of $C_x - B_x$ i.e. the RHS value for eqn 1. Let's say the agent wants to store the following equation into the knowledge base:

$$A + B + C + D + E = 3 \tag{4.7}$$

Then, the equation is stored as the following list:

$$[\{A, B, C, D, E\}, 3] \tag{4.8}$$

The knowledge base is in turn a list of such equations. Hence, in agent 4, knowledge base is a 2 dimensional list that stores equations. Other information that Agent 4 stores to draw inferences is as follows:

1. **class Agent4**
    - agent_grid: 2-D array that stores the current knowledge of the agent.
    - grid: 2-D array that stores the full knowledge of the grid.
2. **class InferenceAgent4**
    - agentGridObject: Object that represents the grid of the agent

- kb_eqns: The list that stores all the equations using which the agent will make inferences
- q: dequeue that stores the equations that the agent will process

3. **class GridWorld**:

- grid: 2-D array that represents the full knowledge of the grid
- sensed_x: Python built-in set that stores cells(as tuples) that has already been visited.
- status_x: Dictionary that stores the status of all the visited or inferred cells or cells that are present in the equations of knowledge base. The dictionary contains key-value pair where the key is the cell (represented by a tuple) and the value is one of the following values: -1 for unconfirmed cell, 0 for unblocked cell and 1 for blocked cell.

4. **class MyPriorityQueue**

- **current_heuristic**: value that determines the metric using which the priority queue will calculate priority. If the value is 'm', the heuristic used will be Manhattan Distance, if the value is 'e', then the heuristic used will be Euclidean Distance and if the value is 'c' then the heuristic used will be Chebyshev Distance.
- **_data**: Sorted Set that stores tuples. Used to implement Priority queue.
- **g**: Used to store the distance of the current cell to the source vertex.
- **h**: Used to estimate the total distance from the current cell to target cell.

4.3 Design

Initially, the agent is present on (0,0) cell and the target is (100,100). Under the free space assumption, initially the agent_grid is empty and the agent will find a path through the agent_grid using A-star algorithm. After getting planned path from A-star algorithm, the agent starts traversing the grid. After every cell, the agent runs inference engine that tries to draw inferences from new knowledge drawn from traversing the grid. The agent obtains two information when going to the next cell x:

- The status of the cell in which the agent is entering.
- The value of C_x from which the agent can create a new equation from the neighbourhood of cell x

Using the value of C_x, the agent can form an equation: $A_1 + ... + A_{H_x} = C_x - B_x$. Now to explain different techniques inference engine applies to draw inferences, lets consider an equation, that is currently not present in the knowledge base. Also the knowledge base contains equations that the agent has already added while traversing previous cells. Let's assume that we have an equation from knowledge base called **kb_eqn** and the agent obtains a new equation from the current cell called **q_eqn**.

4.3.1 Simplification using Subsequence

Let's consider what happens when one equation is sub-sequence of another. If one equation is subsequence of another, this implies that the value of the shorter equation can be used to reduce the length of the longer equation. This technique gives rise to two cases:

(a) **kb_eqn is a subsequence of q_eqn**
Since **kb_eqn** is a subsequence of **q_eqn**, we can reduce the equation **q_eqn** using **kb_eqn**. Since each equation contains a set and an integer, we can check whether the LHS of an **kb_eqn** is a subsequence of LHS of **q_eqn** by checking whether the set in **kb_eqn** is a subset of set in **q_eqn**.

(b) **q_eqn is a subsequence of kb_eqn**
Similar to the above case, we can reduce the **kb_eqn**. To check whether **q_eqn** is a subsequence of **kb_eqn**, it is equivalent to see whether set in **kb_eqn** is a subset of set in **q_eqn**.

For example, consider that the Knowledge Base of the agent contains the following **kb_eqn**:

$$B + C = 1 \tag{4.9}$$

and, when the agent travels to the next cell, it forms the following **q_eqn** equation:

$$B + C + D + E + F + G = 3 \tag{4.10}$$

From above, one can see that equation 4.9 can reduce equation 4.10 and hence we can simplify equation (4.10) to:

$$D + E + F + G = 2 \tag{4.11}$$

It is useful to reduce the length of the Left Hand Side of the equation since it helps to keep the equations simple. Also less variables implies that we can easily solve the equations.

4.3.2 Simplifying using Set Difference

In this section, the agent tries to draw inference by subtracting two equations. Without the loss of generalisation, let's say that the RHS of **kb_eqn** is greater than or equal to the RHS of **q_eqn**. When the agent subtracts **q_eqn** from **kb_eqn**, we can get the equation of the form:

$$\sum_{i=1:r} kb\_eqn_i - \sum_{i=1:s} q\_eqn_i = RHS_{kb} - RHS_q \tag{4.12}$$

From our assumption, the RHS of equation (4.12) is greater than or equal to 0.r represents the number of variables with positive signs in the equation. s represents number of variables with negative signs in the equation. Now the agent wil be able to solve the above equation, if $r = RHS_{kb} - RHS_q$. The other two cases are not solvable wth this information since:

- If $r > RHS_{kb} - RHS_q$, then we have one or more combination in the variables that satisfies the equation. Let's keep the all $q\_eqn_i = 0$, and then we can choose $\binom{r}{t}$ ways to solve the term, $\sum_{i=1:r} kb\_eqn_i = t$, where t = $RHS_{kb} - RHS_q$.
- If $r < RHS_{kb} - RHS_q$, then there is no solution, since there are not enough term swith positive terms to reach RHS of equation (4.12).

For the case of $r = RHS_{kb} - RHS_q$, one can say that the only solution for this case is each variable $kb\_eqn_i = 1$ and $q\_eqn_i = 0$. This is because if any one variable in $kb\_eqn_i = 0$, then there are not enough terms to get the sum $RHS_{kb} - RHS_q$, so all the variables $kb\_eqn_i = 1$. Also, if all the terms kb_eqn_i = 1, then we get $\sum_{i=1:s} q\_eqn_i = 0$. From the summation, we can infer that variables $q\_eqn_i = 0$.

So, from the above discussion, we can say that if we subtract two equations such that the RHS is non-negative and the number of terms with positive sign is equal to RHS, then all the terms with positive signs are 1 ***and all the terms with negative signs are*** 0***.***

4.3.3 Simplifications adopted from Agent 3

In agent 3, the inference engine uses the following two equations to solve the inferred cells:

- $C_x = B_x$: This condition denotes that we have inferred all the blocked cells present in the neighbourhood, and hence all the hidden neighbours in the neighbourhood of x are empty.
- $N_x - C_x = E_x$: This condition denotes that we have inferred all the empty cells in the neighbourhood of x, hence all the hidden neighbours are blocked.

The equation $C_x = B_x$ will arise when RHS of the equation of agent 4 is 0 (because $RHS = C_x - B_x$). Also, if the RHS = 0, we can easily see that if atleast one of the hidden variable is 1, it will contradict with the knowledge base equation. So, the first knowledge base condition of Agent 3 can be incorporated in Agent 4 as follows:

If the RHS of any equation is 0, then we can infer that all the variables of that equations are zero i.e. all the hidden cells are unblocked.

The equation $N_x - C_x = E_x$ will arise when we have confirmed all the empty cells in the neighbourhood of cell x. When we have confirmed all the empty cells in the neighbourhood of x, will mean that all the hidden variables are blocked. The Knowledge base will contain an equation with these hidden variables. In that equation, since all the hidden cells will be blocked, it can be seen that the equation will have the number of variables same as the value in RHS of the equation. Also, in the previous subsection, it has been shown that if number of terms with positive signs is equal to the value in RHS, then all the terms with positive signs has to be 1. In this case, all the terms in the Left Hand side of

the equation will evaluate to be 1. So, the second knowledge condition of Agent 3 can be incorporated as follows:

If the RHS of any equation is equal to the number of terms with positive sign in the LHS, then we can infer that all the variables with positive sign of that equations are one i.e. all the hidden cells are blocked.

4.3.4 Inference Capacity of Agent 4

Does Agent 4 infer "everything that is possible to infer" and if not, why not? Can you construct situations where inference is possible, but Agent 4 doesn't infer anything?
In our implementation, Agent 4 infers everything that it can and we can not find a situation where Agent 4 can not infer something that can be inferred. If we put a time cap on our inference engine instance then shallow inferences, like those in Agent 3, are inferred and deep logical inferences are missed and thus there is less improvement in Agent 4 as compared to Agent 3 performance in terms of time taken, path length, trajectory length and number of bumps if we interrupt the inference engine instance in between.

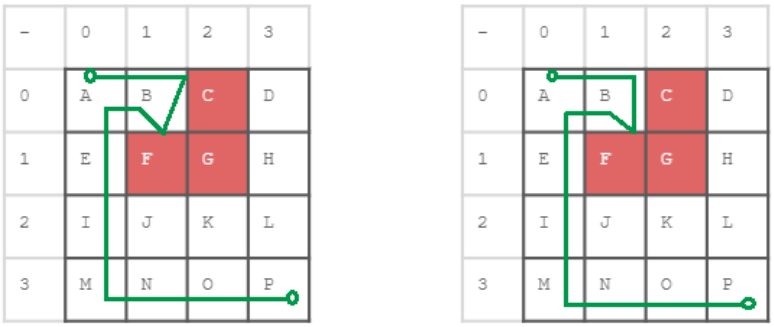
4.4 Difference Between Agent3 and Agent4

The most important difference between Agent 3 and Agent 4 is that the former infers only from the equations that contain hidden cells of a particular neighbourhood of the cell, whereas agent 4 combines different equations collected from all the neighbourhoods and takes possible inferences from these equations. Consider the case, where we are taking the difference between the two equations. Let's take:

1. A+B+C+D = 3
2. C+D+E+F = 1 ; When we subtract Eqn2 from Eqn1, we get:
3. A+B-E-F = 2

From eqn 3, we can infer that A = 1, B = 1, E = 0, F = 0 and using this we can make more inferences in agent 4. But agent 3 will not make any inferences when it has this same knowledge.

Another way in which we can make more inferences is using method of substitution. Now, we will consider a grid in which Agent 4 has less number of bumps than Agent 3 since Agent 4 is able to use the **Simplification using Subsequence** to infer more cells. Consider the following grid as in Figure 4.1 and let's name the cells as follows:



(a) (b)

Figure 4.1: (a) Agent3 vs (b) Agent4 for number of bumps

(0,0) –> A (1,0) –> D
(0,1) –> B (1,1) –> E
(0,2) –> C (1,2) –> F

In case of Agent 3, first planned path is formed using A*, A → B → C → .... → Target

State	C_x	H_x	E_x	B_x	Inference Result
Currently in A	1	3	0	0	A is unblocked
Currently in B	3	4	1	0	B is unblocked
Currently in C	NA	4	1	0	C is blocked

From the above table, we can see that agent will go from A to B to C, but it won't infer any block at cell C. So, it will bump at cell C. Since, C is blocked so the agent will re-plan using the A-star algorithm and find the path as (0,1) to (1,1) to Goal. Here, it will bump again in cell (0,1). Then, when the agent again re-plans, it will not bump at any cell and it will reach the target. So, in this grid, agent 3 will get bumped twice.

Now, considering agent 4:

1. when the current cell is in A, we get: B+D+E = 1; A = 0
2. When the current cell is in B we get: B = 0; A+D+E+F+C = 3

Now in agent 4, we know that B+D+E = 1 and we can infer D+E = 1 by applying B=0. Also, we have A = 0 and B = 0 and D + E = 1, we can put these values to get F + C = 2 from (2), which implies F = 1 and C = 1. So, when we are at B, we are inferring C = 1, which saves us from one bump. After replanning from cell B, agent 4 will again bump at E and then in the next A-star cycle, agent 4 will reach the target.

Also, from the graphs in Section 4.6, we can see that as p increases the performance of Agent 4 gets significantly better than Agent 3. We can understand this observation by understanding that the agents follow free space assumption.

When p is less, the agent 3 and agent 4 plan the path by assuming unconfirmed cells to be unblocked cells. The probability of cells being unblocked is higher since p is less. So, the free space assumption does not conflict much at less p. But as p increases, the free space assumption contradicts more, which depicts the importance of inference agent by having less bumps.

4.5 Agent3 vs Agent4 Inference Capacity

Agent 3 does inference using following equations:

(a) If $C_x = B_x$ implies that all remaining hidden neighbours of x are empty.
(b) If $N_x - C_x = E_x$ implies that all remaining hidden neighbours are blocked.
(c) If $H_x = 0$ implies nothing remains to be inferred about cell x.

Considering the implementation of Agent 4, we can infer that it does all the inferences done by agent 3 (Discussed in detail in section 4.3.3). Consider these aspects of Agent 4:

1. The first knowledge base condition is $C_x = B_x$. If the number of blocked neighbours is equal to the number of confirmed blocked neighbours, then we can infer that the rest of the blocks are empty. This is solved in agent 4 by using the method of subsequence: Let's say A, B, C, D, E, F, G, H neighbours and we have confirmed A, B, C, D to be blocked. And we know that the total number of blocked neighbours are 4. Hence, we can write:

   - A + B + C + D + E + F + G + H = 4
   - A = B = C = D = 1
   - Using above two equations, we get E + F + G + H = 0

   Agent 4 will infer this last equation to be E = F = G = H = 0.

2. The second knowledge base condition is $N_x - C_x = E_x$. This condition means that when $N_x - C_x$, i.e. the total number of unblocked cell is same as the confirmed empty cell, then all the unconfirmed cells are unblocked. This is also solved by agent 4 using the method of subsequence. Let's take another example and say A, B, C, D, E, F, G, H neighbours and we have confirmed A, B, C to be unblocked. Let's say we have $C_x = 5$ and $E_x = N_x - C_x = 8 - 5 = 3$. Then our equation for agent 4 will become:

   - A + B + C + D + E + F + G + H = 5
   - A = 0, B = 0, C = 0
   - Hence, D + E + F + G + H = 5

   Since, number of variables is equal to the RHS, then agent 4 concludes that D = 1, E = 1, F = 1, G = 1, G = 1, H = 1. Hence, agent 4 has the same inference as agent 3 in this case.

3. $H_x = 0$. In this case nothing is required to infer from the cell. In the case of Agent 4, this situation is the same as 0 = 0, where there are no variables in LHS and RHS is 0.

Hence, we can conclude that, agent 4 will always infer what agent 3 does.

4.6 Implementation

The agent starts traversing the graph from (0,0). First, it runs A-star algorithm to plan a path from (0,0) to the target (100, 100). Then, the agent starts traversing the planned path and at each step, the agent infers two things:

- Whether the current cell is blocked or not.
- If the current cell is blocked, the agent senses the value of C_x, which represents the number of blocked neighbours at cell x.

After collecting the information, the agent runs the **infer** function.

From the first information, the agent only marks the current cell as blocked or unblocked. From the second information, the agent will create an equation from it's current knowledge. In this section, the implementation of inference engine for Agent 4 is explained.

4.6.1 Queue Data Structure

Before jumping into the pseudocode, one needs to understand how and why are we using queue and when we are adding a new equation in the queue.

The summary of the workflow of our inference engine is as follows:

1. Assume initially some equations are present in the knowledge base and the agent went to the next step of planned path.
2. The agent senses C_x and it creates a new equation containing hidden neighbours of the cell.
3. The inference agent uses this equation to simplify the knowledge base equations or uses knowledge base equations to simplify the new equation.
4. If the agent is using simplification using subsequence, then we will add the following equations to the queue:
   (a) If any knowledge base equation is simplified, then the new knowledge base equation should be added to the queue. This is because it is important to check whether we can simplify other equations in knowledge base using this simplified knowledge base equation.
   (b) If the new equation is simplified, then we need to again compare the new simplified equation with all of the equations in knowledge base to check whether any other equation is simplified or not. So, if the new equation, which is the reduced equation of q, is simplified, we need to add the reduced equation to the front of the queue.

5. Let's see what will happen if the inference agent simplifies using set difference method:
   Consider the following example, we have the knowledge base equation - A + B + C + D + E = 3 and we have the equation from queue as C + D + E + F + G = 1. Now when we subtract the two equations, then we get A + B - F - G = 2. This implies A = 1, B = 1, F = 0, G = 0. But we also get the equation, C + D + E = 1, which we will push into the queue.

Also, we need to discuss when will we add the new equation that the agent gets when traversing the planned path. Consider the following cases:

1. Case 1: No simplification occurs using sub-sequence or set difference method - The new equation will be added to the knowledge base.

2. Case 2: Simplification occurs using the set difference method - When simplification occurs through the set difference method, we will infer all the nodes that are not present in both the knowledge base equation and queue equation (equation popped from queue). For the cells that are present in both these equations will form another equation, which is already present in knowledge base. It is already present in knowledge base since we can put the inferred values to the knowledge base equation and get the equation that contains common cells. So, in this case it is not required to add queue equation to knowledge base.

3. Case 3: Simplification occurs using Method of Subsequence: Here two things can happen: either knowledge base equation is subsequence of queue equation or vice versa:

   (a) **Knowledge Base Equation is subsequence of Queue Equation:**
   This means that using method of subsequence, only the equation we get from the queue will be reduced using Knowledge base equation. Now, the knowledge base equation divides the queue equation into two parts: one part is the same as knowledge base equation and another part is reduced queue equation that we are already pushing in the queue. Hence, we are retaining all the information of queue equation. So, it is not required to add queue equation to knowledge base.

   (b) **Queue Equation is subsequence of Knowledge Base Equation:** This means using the method of substitution, we are reducing the knowledge base equation using queue equation. So, we are removing the old knowledge base equations and adding new knowledge base equations in the queue. This has no effect on queue equation and hence in this case we will add queue equation to the knowledge base.

Seeing the above cases, we concluded that whenever queue equation is simplified we will not add the queue equation to the knowledge base and if the queue equation is not simplified we will add queue equation to the knowledge base.

4.6.2 Pseudocode

TRAVERSE_PLANNED_PATH(planned_path):

1. Traverse through each cell of the planned path. For each cell do:
   (a) cell ← planned_path[i], currx ← row_index , curry ← col_index
   (b) If the cell is not visited by the agent do:
      i. Mark the cell as block or empty in the agent grid based on the grid containing full knowledge. Also, update status_x[cell] to 1 or 0 depending on whether the cell is blocked or not.
      ii. PARTIAL_SENSING() function is called that calculates the number of neighbours of **cell** that are blocked. Update the value of C_x for the cell in cx dictionary. This is called only if the **cell** that is marked is empty.
      iii. Call INFER() function that starts inferring from the **cell**
   (c) If cell is visited, then fetch the *curr_status* of the **cell** from the current knowledge of the agent.
   (d) If the current cell is blocked, then set the **restart_cell** to the previous cell of the planned path. **restart_cell** is the cell from which the Repeated A-star will start replanning. Break from the loop that traverses from the planned path.
   (e) Check whether the next four cell of the planned path are inferred to be block.
   If cells in the next four steps are inferred to be block, then set the **restart_cell** to the **cell**, append the current cell to **traversed_path** and return the traversed_path and the **restart_cell**
   If the cells in the next four seps are not blocked, then append the **cell** to the traversed_path and the agent will move on to the next cell.
2. Once the agent traverses the full planned_path, just return the traversed_path and restart_cell.

PUT_VAL_KB()
This function checks for each equation in kb_eqns(list of all knowledge base equations known to agent) and updates each equation according to the current knowledge of the agent. The new/reduced equation is removed from the kb_eqns(knowledge base) and added in the queue.

1. Traverse through the equations present in the knowledge base. For each equation, do the following:
   (a) Get the set containing LHS part of the equation in *var* and store the value of the RHS part of the equation in *val*. Create a new set to store the new equation called *new_eqn*. Traverse through each cell of the set and do:
      i. Check whether the status_x of the cell is 0 or 1 i.e. whether the agent knows that the cell is blocked or not. If yes, change the value of *val* to $val - 1$ if the cell is blocked and do not change the *val* if the known cell is unblocked.

ii. If the status_x of the cell is not known i.e. the agent has not yet inferred the cell, then add the cell in the *new_eqn* set.

(b) If any cell present in the equation is known i.e. the agent is able to reduce the equation based on it's current knowledge then remove the equation from the knowledge base and add the *new_eqn* to the queue.

GET_EQN_CURRENT_CELL(cell)
This function is used to generate an equation for a cell, according to the current knowledge of the agent.

1. Create a new set called *var* to store the LHS part of the equation. Initialise the integer *val* to $C_x[cell]$ to store the RHS part of the equation.
2. Traverse to every valid neighbour of cell and do the following:
   (a) If the neighbour of the cell is not present in status_x (which means that this variable is not known and is not present in knowledge base) or if it is present then the value of the cell in status_x is "-1", then add the neighbour to the new set (since the neighbour is hidden)
   (b) If the neighbour is known to be blocked just reduce the *val* by 1. Don't do anything if the neighbour is not blocked.
3. return *var*,*val*

SIMPLIFY_EQUATION(eqn):
This function simplifies the equation, according to the current knowledge of the grid.

1. Create a set called *var* to store the LHS part of the equation and create an integer variable to store *val* that represents the RHS part of the equation.
2. For each cell in the eqn do the following:
   (a) if the cell is known to be blocked or unblocked in the current knowledge of the agent, then decrease the value of *val* by 1 if the cell is blocked. Don't do anything if the cell is known to be unblocked.
   (b) if the cell is not known, then add the tuple of the cell in the set called *new_eqn*
3. return [new_eqn, val]

CHECK_SOLVABILITY(eqn):
This function checks whether the current equation is solvable or not. There are two cases:

- If number of variables == val → then the value of all variables is "1".
- if RHS == 0 → then the value of all variables in the eqn is 0

Algorithm

1. Fetch the *var* and *val* from the *eqn* argument.
2. Check the two conditions: the len(var) == val or val == 0:
   (a) If the len(var) == val, then we need to set all the cells in the *var* to be blocked. Traverse the cells in the eqn and update the grid information and status_x to be blocked. Set the $flag$ to True.
   (b) If the val == 0, then we need to set all the cells in the *var* to be empty. Traverse the cells in the eqn and update the grid information and status_x to be empty. Set the $flag$ to True.
3. return $flag$

SIMPLIFY_KB_SUBSEQUENCE(q_eqn):
This algorithm implements the technique of *Simplification by Subsequence* as explained in the Design Section of Agent 4. We need to check whether the equation can be simplified using knowledge base equations or it can simplify any knowledge base equations.

1. Create a variable **flag** that indicates whether the input equation will be added to the knowledge base or not. Initially, **flag** is set to True. Traverse over all the equations present in the knowledge base and do the following: (Each equation in Knowledge base is called kb_eqn)
   (a) Fetch **kb_var** and **kb_val** from each knowledge base equation. Also, fetch **q_var** and **q_val** from **q_eqn**
   (b) If the equation from the queue is same as knowledge based equation, then set **flag** to False and break out of the loop, since it is a redundant information.
   (c) res = CHECK_SUBSEQUENCE(kb_var, q_var). **res** is True if **kb_var** is subsequence of **q_var**.
      i. If res is True, then remove the **kb_eqn** from the knowledge base. Create a new set called **new_var** and a new integer variable called **new_val** to store the new reduced equation
      ii. **new_var** is calculated as the set difference between the **kb_var** and **q_var** and the **new_val** will be the difference between the **val_kb** and **val_q**
      iii. If the length of **var_kb** is 1, then we are not required to further solve the equation and we will get the inferrence of the single cell accordingly.
      iv. If the length of **new_var** is greater than 1, then we need to append the equation to the queue q.
   (d) res = CHECK_SUBSEQUENCE(q_var, kb_var). **res** is True if **q_var** is subsequence of **kb_var**
      i. if res == True, set $flag$ to False. Also, create a new equation by taking the set difference between **q_var** and **kb_var**. Also, calculate the new value of RHS by taking the difference between the **val_q** and **val_kb**.

2. After simplifying all the equations in the knowledge base using this technique, we call PUT_VAL_KB() to update the knowledge base equations according to the current knowledge of the grid.

3. return flag

SIMPLIFY_KB_SET_DIFF(eqn):

1. Create a variable called **flag** that indicates whether the **eqn** will be added to queue or not. Traverse through all the equations in the knowledge base and for each equation extract the LHS and RHS part of the knowledge base equations and **eqn**. For the knowledge base equation, let's call the LHS part to be **kb_var** and RHS part to be **kb_val**. For the **eqn**, lets call the LHS part to be **q_var** and the RHS part to be **q_val**. Then for each equation in the knowledge base do the following:

   (a) Take the set difference of the kb_var and q_var such that the difference of the corresponding RHS values is non-negative. Let the output of set difference be **set_diff** and difference of RHS is **val_diff**.

   (b) If the length of **set_diff** = **val_diff**, i.e. the number of elements with positive sign is equal to the RHS value, then the value of all the variables with positive sign is "1" and the value of all the variables with negative sign is "0". Also, set the value of **flag** to be False.

      i. Since, we already have the variables with positive sign in **set_diff**, we can iterate over the cells and mark them as blocked in knowledge of the agent.

      ii. And, to get the cells with negative signs, we do the difference of sets such that the RHS is now negative. Let's store this new set difference in **set_diff2**. Then the inference engine iterates over this **set_diff2** and mark all the cells present in this set as empty.

2. PUT_VAL_KB() is called to update the equations in knowledge base according to the current knowledge of the agent.

INFER(type_, cell)

1. Calls PUT_VAL_KB()
   Comment: Update the knowledge based eqns based on current knowledge of the grid

2. Calls GET_EQN_CURRENT_CELL(cell). Returns *var*,*val*
   Comment: Generates an equation containing hidden cells of the neighbourhood.

3. Append the $[var, val]$ to the queue q.

4. Iterate over all the equations of queue until the queue gets empty:

   (a) Pop an equation from the queue. Let's call it *eqn*.

   (b) Call SIMPLIFY_EQUATION(eqn)

(c) $flag$ = CHECK_SOLVABILITY(eqn). If the equation is solved then it is not required to add it to the knowledge base. Hence, in the next step, we just update the knowledge base based on the updated knowledge

(d) If flag is True, call PUT_VAL_KB()

(e) add_to_KB1 = SIMPLIFY_KB_SUBSEQUENCE(eqn)

(f) add_to_KB2 = SIMPLIFY_KB_SET_DIFF(eqn)

4.7 Analysis

4.7.1 Number of Bumps

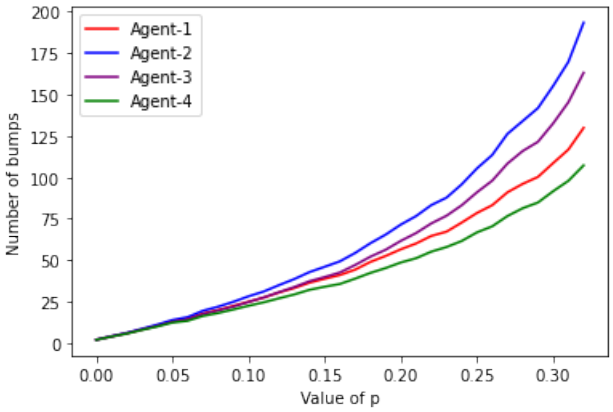
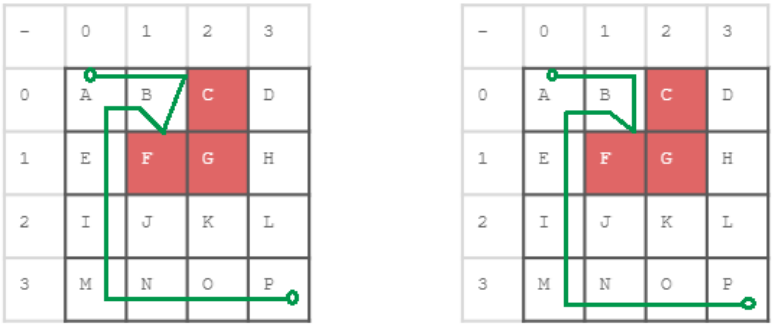


Figure 4.2: Number of Bumps vs Density

Though Agent 1 has concrete information about its neighbourhood, it does not infer the cell states of far neighbours and thus gets blocked deep in the



(a) (b)

Figure 4.3: (a) Agent3 vs (b) Agent4 for number of bumps

planned path whereas agent 4 prunes the current planned path early if it infers a block down the path and thus does not bump.

Also, Agent 2 is blindfolded and thus has the least amount of information available and therefore performs the worst in terms of blocks encountered.

Even though Agent 4 may be able to infer more things - do those situations ever actually arise in practice? As Agent 4 can infer more information about its environment, the agent 4 grid is closer to the known grid than the agent 3 grid. Therefore, agent 4 plans around a truer grid with less free space assuming(hidden) cells and thus agent 4 plans better and encounters less number of blocks as a result. For Example: In *Figure 4.3: (a) Agent3 vs (b) Agent4 for number of bumps*, Agent 3 bumps twice whereas Agent 4 bumps only once saving time and processing power. One such practical situation can be a robot trying to traverse through a hazy or stormy environment where vision is blurred and sensors are able to give partial information.

4.7.2 Number of Cells Processed

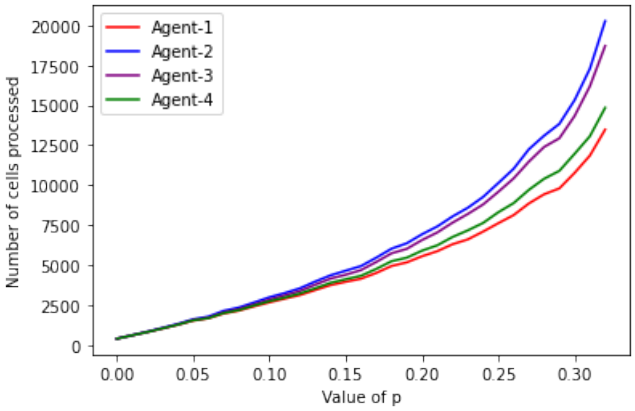


Figure 4.4: Number of Cells Processed vs Density

The number of cells that needs to be processed for the agent to reach the goal for different agents at increasing block densities is the least for Agent 1 which is slightly less than that for Agent 4. This is because Agent 4 does not have much information for inference initially and thus explores more cells but as it traverses and gets more concrete information using inference and bumping, the environment is intelligently traversed.

As we figured, Agent 2 has the least information about the environment and thus needs to roam around more to gather quality information to reach the goal. Therefore the number of cells processed for Agent 2 is the highest.

As Agent 4 can infer more information about its environment than agent 3, the agent 4 grid is closer to the *known grid* than the agent 3 grid. Therefore, agent 4 plans around a truer grid with less free space assuming(hidden) cells and thus Agent 4 plans better and thus there is less backtracking as it often finds blocks far away in the planned path. And as a result, the number of cells that are actually processed is less.

4.7.3 Number of Discovered Cells

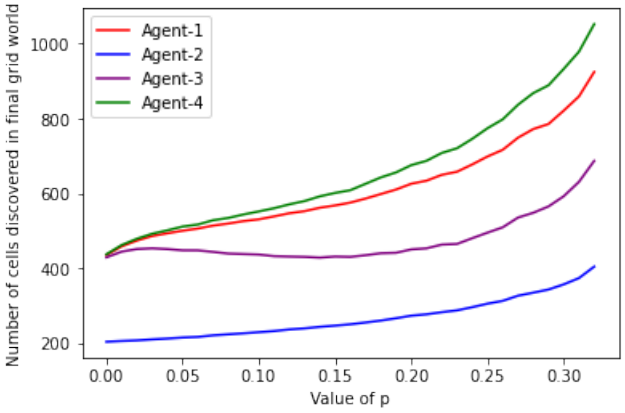


Figure 4.5: Number of Discovered Cells vs Density

As expected, the number of cells inferred by agent 4 in the final discovered gridworld is significantly more than the number of cells inferred by other agents. The ability of Agent 4 to combines individual inferences to make new inferences outperforms not only agent 3 inference but also the definitive knowledge of agent 1 as Agent 4 can infer information about far away cells and thus is capable of discovering more cells. The only cells Agent 2 could have inferred are the cells along its trajectory and hence it has the least number of cells discovered in its trajectory.

4.7.4 Trajectory Length

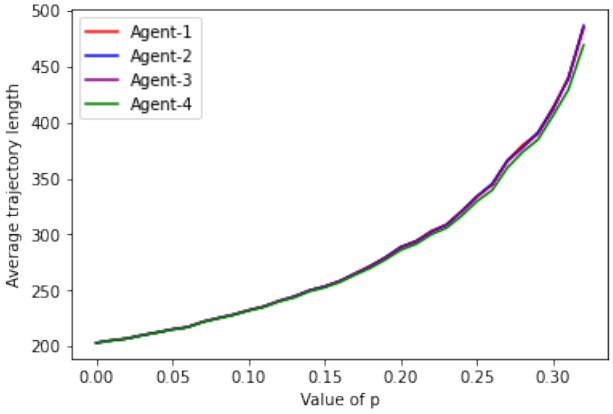


Figure 4.6: Trajectory Length vs Density

This metric is thought-provoking as it is the only metric on which all the agents perform alike without much variation despite them having different level of sensing and inference capabilities. Though it is not clear from the above

graph, but Agent 4 has less trajectory length than Agent 1, 2 and 3 when we look closely at the analysis data. This is because of a special inference ability of agent 4 and 3 that checks for blocks four steps down the planned path and thus less backtracking happens and as a result, trajectory length is less as compared with other agents since agent 4 infers more knowledge about the far neighbors; the possibility of encountering blocks 4 steps down the path is more as compared to Agent 3.

4.7.5 Path Length

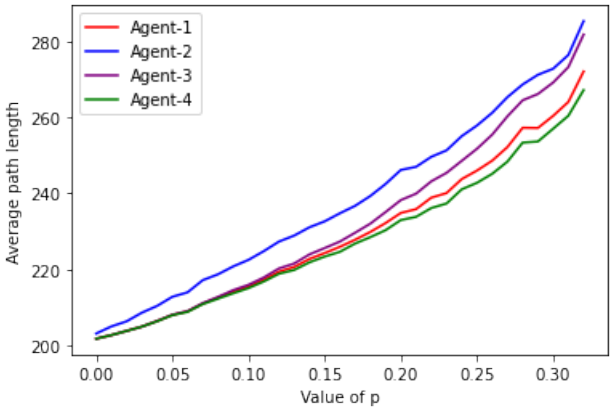


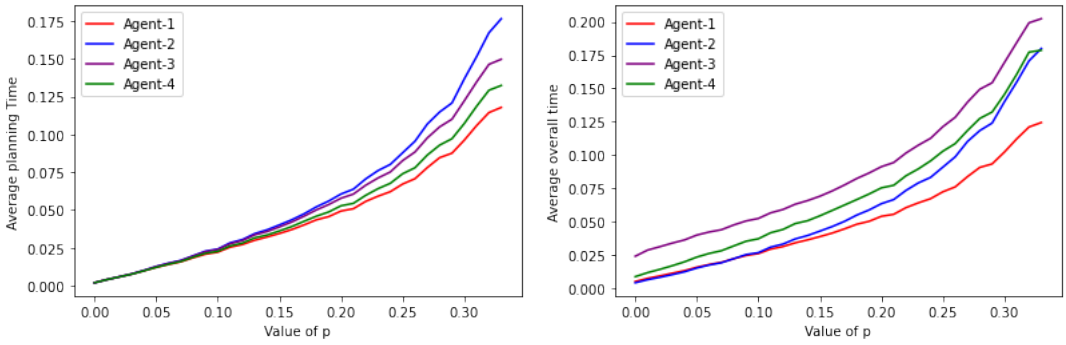
Figure 4.7: Path Length vs Density

The shortest path travelled by the agent on *Final Discovered Grid* is Path Length. We observe that the Path Length for Agent 4 is less than other agents. We also know from *Figure 4.5: Number of Discovered Cells vs Density* that Agent 4 has the highest discovered cells in Final Discovered gridworld. Thus Agent 4 has access to more information about the *Known Grid*. Thus less of the available paths are considered blocked in the final discovered gridworld and thus the agent is able to find shorter path to the goal. *Quality information available to Agent 4> Agent 1> Agent 3> Agent 2*

4.7.6 Planning Time and Overall Time

From *Figure 4.4: Number of Cells Processed vs Density* we can see that Agent 1 processes the least number of cells and also has the least number of bumps following Agent 4, Agent 3 and Agent 2. This implies that Agent 1 has to process less cells during the planning phase and it also has to replan less number of times which reduces its planning time and the same for other agents. Agent 4 still ranks second on this metric. Next we observe that Agent 4 ranks third on overall time taken. This is because Agent 4 needs to make inferences while Agent 1 and 2 has to make no inferences.

Agent 4 performs better than Agent 3 here even though it infers more information than Agent 3. This is because due to more access to the concrete information, Agent 4 cuts down on the amount of planning time and thus compensating for spending time doing more inference and as a result beating Agent



(a) (b)

Figure 4.8: (a) Planning Time vs Density (b) Overall Time vs Density

3. It is also worth noting that the inference engines of agent 3 and 4 are different from each other and thus even though Agent 4 does more inference, it does not spend a drastically more time for the same as we have optimized Agent 4 inference engine(equation solving) for its specific characteristics.

4.7.7 Number of Early Terminations

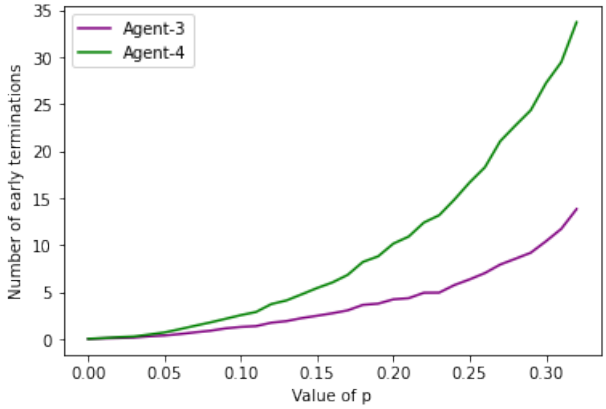


Figure 4.9: Number of Early Terminations vs Density

As agent 4 has better inference capabilities, it can infer the cell states of near and far neighbors and thus find blocks in the planned path earlier that Agent 3 thus terminating blocked path. This saves time and processing power that might have been used to traverse a useless path and instead Agent 4 uses these resources to find a better path.

4.8 Computational Issues

1. **Knowledge Base storage space**

   - When we get a new equation comprising knowledge of neighbours of a particular node; we try to solve other existing equations in our knowledge base to infer any direct information about the status of a node being blocked or unblocked. There will be multiple equations each consisting maximum 8 variables (neighbours of a node); hence there will be too many variables. We can make a new equation when we solve two equations but all such equations might not be useful for any direct inference and hence the new equation will just represent redundant data and will also occupy additional storage space in our knowledge space. For example:-
     - KB equation: A + B + C + D = 3
     - Q equation: D + E = 1
     - New equation: A + B + C - E = 2

     The above new equation does not give any direct inference and adding that in knowledge base will be redundant and will occupy unnecessary space.
   - **Solution:** We tackle this issue in our inference engine of agent4 by conditionally handling these new equations. We use two methods to solve equations: Subsequence and Set difference. After both the methods are completed then only one iteration is considered to be over. After every iteration we will decide whether to add q equation to kb or not. If q equation is not reduced (Case1 in subsequence method never occurs) and if no equation is solved using set difference (set difference doesn't solve at least one equation) then only we will add our q to our knowledge base.
   - By this way we handle our knowledge base space and also minimizing computational time by preventing any redundant/unnecessary equations in our storage space.

2. **Check equations of knowledge base**

   - We have implemented a function: put_val_kb which makes sure that all equations of our knowledge base don't have any variable which is already solved/inferred. Before popping out a new equation from Queue, this function is called and it will check for each equation in knowledge base if they have any variable whose value was already inferred and stored in our final grid(i.e. check for solvability); then it will reduce that particular equation by putting the respective value from our final grid. In this way we make sure that equations in knowledge base don't have any unnecessary/solved variables and thus making sure all equations are in reduced form possible.

3. **Propagation of information in agent 3**

   - While updating information any cell, we check if the cell is visited or not. If the cell is not visited we will not have Cx value of that cell

and hence we will not propagate the information to its neighbours. In this case we will just update value of Bx, Ex and Hx of that cell and will not propagate information further to its neighbours since we don't have Cx and without that we cannot check knowledge base conditions. If a cell is visited we will update values of Bx, Ex and Hx and we will also have Cx so we can propagate further to its neighbours. By this way we avoid unnecessary propagation of information if a cell is not visited; hence avoiding any unnecessary condition check or adding such neighbours to the queue.

5

Agent 5

5.1 Design

Agent 5 is the agent that uses probabilistic inference to plan the path in the grid. The agent starts from (0,0) and the target is at (100,100). Under the free space assumption, the initial probability of each cell being blocked is 0. The difference between agent 4 and agent 5 is only in the planning phase and the inference engine for both the agents are same. In the planning phase of agent 5, the inference engine calculates the probability for each cell to be blocked based on the knowledge base.
The main idea behind implementing agent 5 is to include probability in planning the path. In our grid world, if we do not know that a cell is blocked, but if we have a significant reason to believe that the cell is blocked, then the agent could avoid that cell.
The agent initially plans a path and the agent calls inference engine after every next step in the planned path. If the agent is blocked by a cell in it's planned path, then the agent will re-plan the path, based on the current knowledge of the grid.

5.1.1 Probabilistic Planning Algorithm

The path planning algorithm for agent 5 will follow the same A-star algorithm in agent 4. In Agent 4, the tuple that was entered in priority queue for cell x was: (f(x), h(x), (x,y)), where

- f(x) = g(x) + h(x), which is the sum of the estimated distance of the cell to the target and the actual distance of the cell from the source cell.
- h(x): represents the estimated distance from the cell to the target

but in agent 5, the tuple that we enter for a particular cell x is: (f(x) + P(x), h(x), (x,y)), where

- P(x): Probability inferred from the neighbour that has maximum confidence for it's neighbours to be blocked.
- f(x), h(x) is same as defined above.

. After the agent traverses a planned path, the agent either reaches a goal or repeats A-star from an unblocked cell. After the agent finishes traversing a planned path, the agent updates probability of each cell. The algorithm used to update probability of each cell is as follows:

1. For each cell in the grid, do the following:
   (a) X is the set of cell in neighbourhood of the cell.
   (b) $i = \text{argmin}_{x \in \mathcal{X}} H(x)$ (Because, argmin returns the argument for which the value of $H(x)$ is least.)
   (c) P(cell) $\leftarrow \frac{C_i - B_i}{H_i}$

The probability that we are calculating in the above step can be interpreted as follows:
For each cell, the neighbourhood has 8 neighbours and we can get the probability of the current cell being blocked from the Knowledge Base equations of the neighbours. Out of all the equations from which we can calculate probability, the randomness will be least for the cell who has the least number of unknowns. The cell who has the least number of hidden neighbours has the most accurate neighbourhood and the probability will also be closer to the actual case.
When the agent runs A-star algorithm for re-planning the path, it creates the tuples for each cell - (f(cell) + P(cell), h(cell), (cell_x,cell_y)). The priority queue compares the values of these tuples to choose the order in which the cell will be processed to plan the path. After the agent plans path using **A-star with probabilities**, the agent travels the planned path and with each next step in the planned path the agent uses inference agent which is same as Agent3.

5.1.2 Graphs and Analysis

Number of Bumps

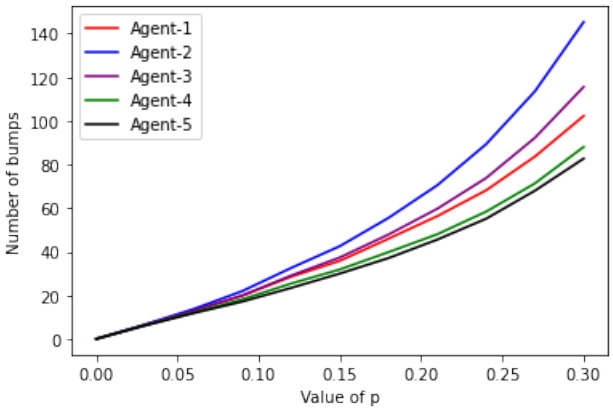


Figure 5.1: Number of Bumps vs Density

Agent 5 makes use of multiple individual inference along with probabilistic information and gets better guidance to reach the goal using an improves heuristic. Thus it plans better using A* and encounters less blocks than the other agents.

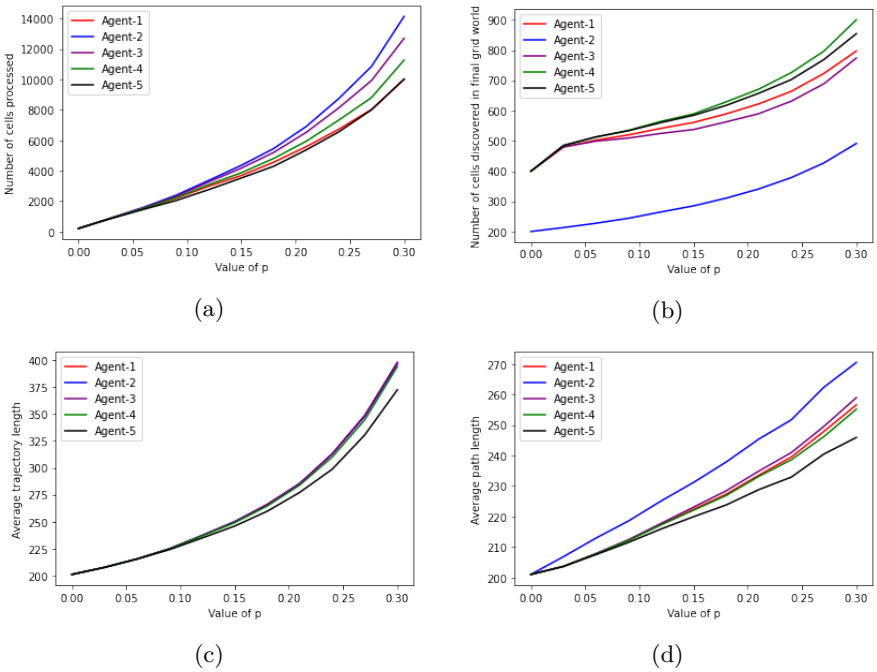


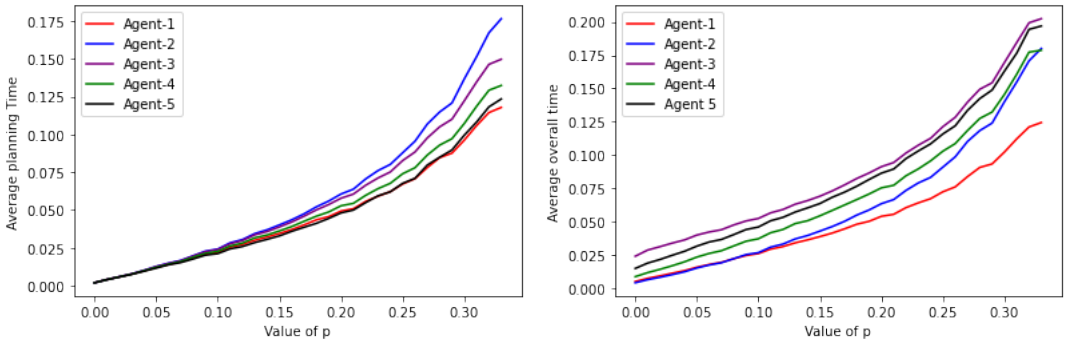
Figure 5.2: (a)Number of Cells Processed vs Density, (b)Number of Discovered Cells vs Density, (c)Trajectory Length vs Density, (d)Path Length vs Density

Number of Cells Processed, Number of Discovered Cells, Trajectory Length, Path Length

We can observe from the above graphs that even though Agent 5 processes and discovers less cells than Agent 4, it performs much better in terms of Path Length and Trajectory Length. This means that Agent 5 chooses right path very early and has to roam around less to get relevant information as it has better guidance due to the utilization of probabilistic information.

Planning Time and Overall Time

Here we can see that as Agent 5 is getting access to better and more information, it has to replan less as it bumps less and thus has to spend less time in the planning phase. But at thse same time it calculates and processes a lot of probabilities which increases the overall time required for the agent 5 to reach the goal,



(a) (b)

Figure 5.3: (a) Planning Time vs Density (b) Overall Time vs Density

5.1.3 Shortcoming of the Approach

The shortcoming of this algorithm is as follows:

1. Since, the probability of a cell depends on it's number of neighbours. Also, we are considering the most accurate probability calculated from the cell with the least number of neighbours. This will create a bias towards the cells in the edges, since the cells near the edges already have least number of neighbours.So, the algorithm might plan poorly in the initial stage by going towards the edges but after a few repeated A-star, the path will be better.

2. In this algorithm, we are not capturing the complete relationship of a neighbourhood of cell x. For a cell x, we are only capturing probability from one cell that most accurately depicts it, but it is better to consider the probability derived from all the neighbourhood to capture the probability.

3. Here, we are doing inference based the parameters calculated in Agent 3. But it should be possible to consider the knowledge based equations in Agent 4. We can combine one or more equations to calculate probability. The method of calculating probability will involve modelling dependency among the variables. This might improve the probability.

4. This algorithm is not optimal since during the literature survey, we found that there is a method of Monte-Carlo that involves finding the probability of taking action based on the rewards and punishment the agent gets when entering a blocked or unblocked cells. Hence, there are other optimal methods than this method.

6

References

- http://proceedings.mlr.press/r4/attias03a/attias03a.pdf