CS 520: Introduction to AI
Wes Cowan

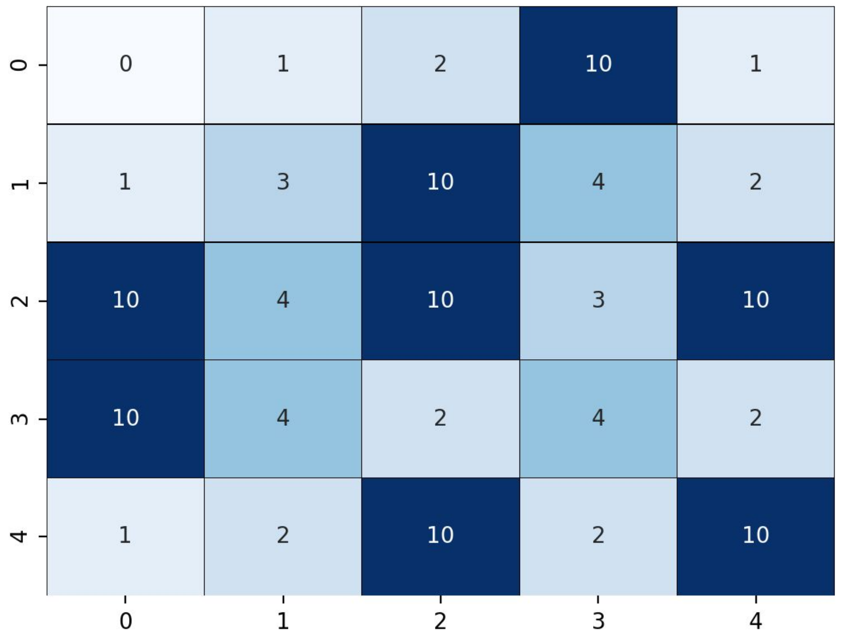
Assignment 2 - MineSweeper, Inference-Informed Action
(October 22, 2019)

197001190: Aditya Lakra (al1247)
196003272: Hari Priya (hp467)
197001859: Twisha Naik (tn268)

TABLE OF CONTENTS

1 Background: MineSweeper

The game of MineSweeper has an underlying board where there are a fixed number of mines spread across the board and all the cells which are not mines describes the number of mines in the surrounding cells.

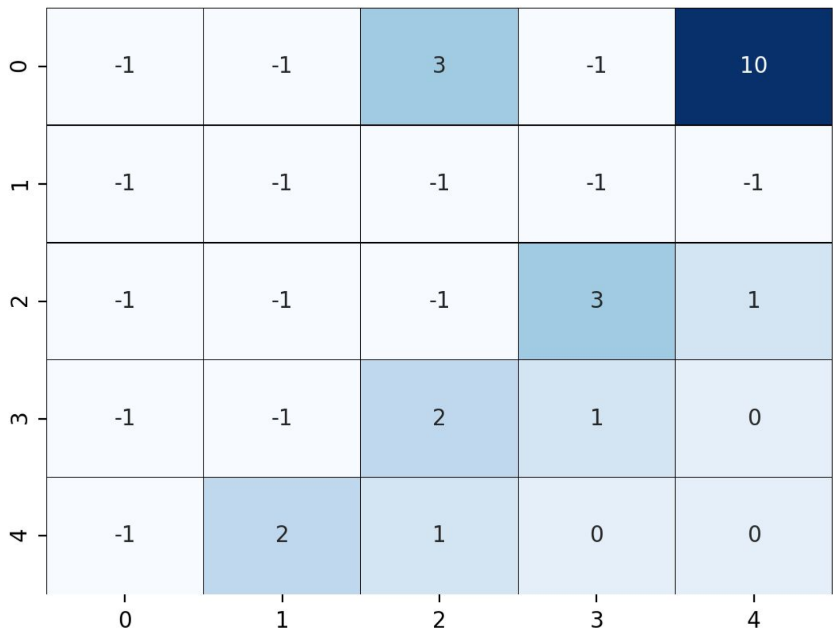


	0	1	2	3	4
0	0	1	2	10	1
1	1	3	10	4	2
2	10	4	10	3	10
3	10	4	2	4	2
4	1	2	10	2	10

Fig: The underlying game board. Here, the value **10 represents a mine** and the other values are the safe cells displaying the number of mines surrounding the cell.

When the game starts, the entire board is unknown to the player. The user randomly opens the cells and the aim is to figure out the locations of all the mines on board. If a user opens a cell and it reveals 1, we know that one of its neighbors is a mine.

Below is the board after few iterations into the game.



	0	1	2	3	4
0	-1	-1	3	-1	10
1	-1	-1	-1	-1	-1
2	-1	-1	-1	3	1
3	-1	-1	2	1	0
4	-1	2	1	0	0

Fig: The board as seen by the agent at a particular point in the game. Here the **-1's are the hidden cells** and the rest of the cells are the explored cells with particular values.

2 Program Specification

2.1 A Basic Agent Algorithm for Comparison

The basic agent algorithm takes into consideration just the local information all the safe cells with clue can infer.

The algorithm is based on the following simple formulae:

- `Number of hidden neighbors = Clue - Number of revealed mines`

  => every hidden neighbor is a mine

- `Number of hidden neighbors =`

  `(Total Neighbors - Clue) - Number of revealed safe neighbors`

  => every hidden neighbor is safe

For implementing this, we just look at the cell which is newly clicked at and we check for the above two conditions for all its neighbors.

2.2 Proposed Algorithm

The algorithm we propose is built upon the baseline approach. Once the baseline algorithm figures out all it can using the local information, we try to combine the information we have from all the open cells on the board and then try to infer more things.

The pseudo code of the overall code is as follows:
* the individual algorithms are explained in detail in the next section

```
Pick an initial cell randomly
Run baseline algorithm

While entire board is not explored:
    While we can make some inference:
        Run basic inference algorithm
        If basic inference algorithm is not able to infer anything:
            Run advanced inference algorithm
        Run baseline algorithm again
```

3 Questions and Writeup

3.1 Representation

How did you represent the board in your program, and how did you represent the information / knowledge that clue cells reveal?

The game board is represented in the form of a two-dimensional array of cell objects.

Local Information Representation

Each cell has the following properties which represent the information or knowledge we have:

- value : Initially initialized to -1.
    - 0 - safe cell
    - 1 - mine cell
    - -1 - covered/unknown cell
- is_mine : Indicates if a cell is a mine or safe. Initially initialized to False.
- clue : Number of mines surrounding the cell(if safe) as indicated by the clue. Initialized to 0.
- safe_cells : Number of safe cells identified around the cell. Initialized to 0.
- mines : Number of mines identified around the cell. Initialized to 0.
- hidden_squares : Number of hidden cells around the cell. Initialized as below when the board is generated.
    - Corner cells : 3
    - Edge cells : 5
    - Remaining cells : 8
- no_of_neighbors : Number of neighbors for the cell.
    - Corner cells : 3
    - Edge cells : 5
    - Remaining cells : 8

Few other properties of cell that are not used as of now, but can be used for the optimization of code.

- set_of_unknown_neighbors : Set of neighbouring cells which are yet to be uncovered. Initially contains all the neighbouring cells.
- set_value : Number of hidden mines around the cell. This value is associated with the set_of_unknown_neighbors so as to infer additional information about covered cells using the inference algorithm. Initialized to -1.
- probability : Probability of the cell being a mine. Initialized to -1.

Global Information Representation Approach-1

Approach 1 is about representation of the knowledge as a property of cell.

Whenever a clue is revealed:

- cell.clue = cell.mines + cell.set_value
- cell.set_of_unknown_neighbors maps to cell.set_value

Now we have a set of potential mines and the total number of cells which are mines. This knowledge needs to be combined with corresponding knowledge of neighboring cells make some inference. One way to reach a conclusion is to reduce the size of the sets in order to make deterministic choices.

Deterministic choices can be made in the following cases:

- Size of (set_of_unknown_neighbors) == set_value => which means all the elements of the set are mines
- Set_value == 0 => which means all the elements of the set are safe cells

Our aim is to reach a deterministic state using set manipulations.

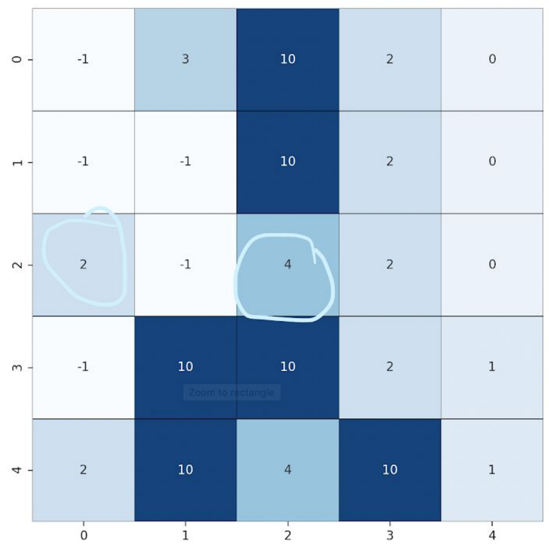
Whenever a cell is queried, all of its neighbors' set_of_unknown_neighbors and potentially set_value would change. Thus, after updating the knowledge of all the neighbors, we try to make further inferences by the following approach:

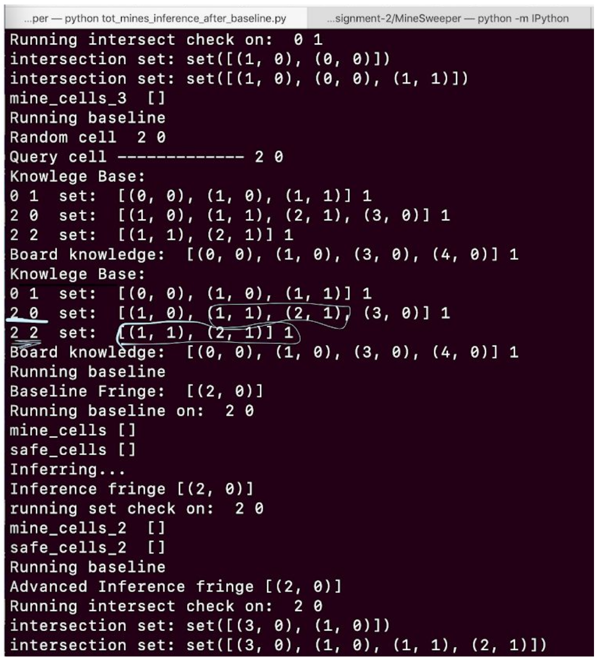
```
While there is an update in any of the neighbor's or the updated cell's set_of_unknown_neighbors:

      check if any of the set is a subset of the other

      If yes:

        subtract the smaller set from the larger

        subtract the corresponding set_values
```

Drawbacks and Failure

We tend to miss out the inferences we could obtain by combining the knowledge of cells which are not immediate neighbors. Consider the following example.

Example




...per — python tot_mines_inference_after_baseline.py
...signment-2/MineSweeper — python -m IPython

```
Running intersect check on:  0 1
intersection set: set([(1, 0), (0, 0)])
intersection set: set([(1, 0), (0, 0), (1, 1)])
mine_cells_3  []
Running baseline
Random cell  2 0
Query cell ------------- 2 0
Knowlege Base:
0 1  set:  [(0, 0), (1, 0), (1, 1)] 1
2 0  set:  [(1, 0), (1, 1), (2, 1), (3, 0)] 1
2 2  set:  [(1, 1), (2, 1)] 1
Board knowledge:  [(0, 0), (1, 0), (3, 0), (4, 0)] 1
Knowlege Base:
0 1  set:  [(0, 0), (1, 0), (1, 1)] 1
2 0  set:  [(1, 0), (1, 1), (2, 1), (3, 0)] 1
2 2  set:  [(1, 1), (2, 1)] 1
Board knowledge:  [(0, 0), (1, 0), (3, 0), (4, 0)] 1
Running baseline
Baseline Fringe:  [(2, 0)]
Running baseline on:  2 0
mine_cells []
safe_cells []
Inferring...
Inference fringe [(2, 0)]
running set check on:  2 0
mine_cells_2  []
safe_cells_2  []
Running baseline
Advanced Inference fringe [(2, 0)]
Running intersect check on:  2 0
intersection set: set([(3, 0), (1, 0)])
intersection set: set([(3, 0), (1, 0), (1, 1), (2, 1)])
```

Over here, if we combine the knowledge of (2,0) and (2,2), we could infer that (1,0) and (3,0) are safe cells. But using the above approach we fail to infer the same.

Potential Corrections:

We can propagate the knowledge that we received to the neighbors of the neighbor to avoid such cases. To implement this, we can follow either a DFS or BFS approach.

To overcome the above mentioned issues we followed the following approach.

Global Information Representation Approach-2

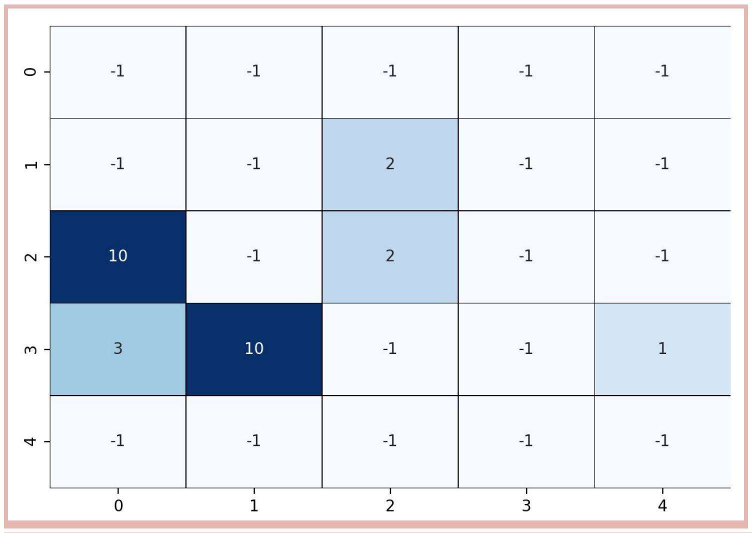
Approach 2 is about representation of a common Knowledge base.

We now have a global knowledge base of the entire board. It is a dictionary with a set of variables as key and the number of mines among those cells as its value.

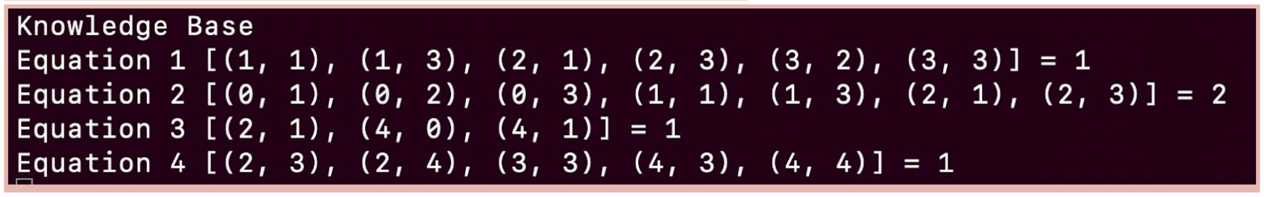
```
Knowledge base dictionary{

    set(cell coordinates): number of mine cells,

    set(cell coordinates): number of mine cells

}
```

Each entry in our knowledge base is an equation with the number of hidden cells as the variables and the value suggesting how many cells out of those are mines.

Example



	0	1	2	3	4
0	-1	-1	-1	-1	-1
1	-1	-1	2	-1	-1
2	10	-1	2	-1	-1
3	3	10	-1	-1	1
4	-1	-1	-1	-1	-1



```
Knowledge Base
Equation 1 [(1, 1), (1, 3), (2, 1), (2, 3), (3, 2), (3, 3)] = 1
Equation 2 [(0, 1), (0, 2), (0, 3), (1, 1), (1, 3), (2, 1), (2, 3)] = 2
Equation 3 [(2, 1), (4, 0), (4, 1)] = 1
Equation 4 [(2, 3), (2, 4), (3, 3), (4, 3), (4, 4)] = 1
```

3.2 Inference

Update of knowledge base

When you collect a new clue, how do you model / process / compute the information you gain from it? i.e., how do you update your current state of knowledge based on that clue?

Whenever a new cell is queried either randomly or by inference or when a cell is marked as mine, we perform the following two things:

a. **Update the information of the cell**

   Change the cell value for the queried cells from the initial value of -1.
   If mine cell, cell.value =1
   If safe cell, cell.value = 0

b. **Update the information of the neighbors**

   Case1: New cell = MINE Cell

```
# for each neighbor reduce the number of hidden squares by 1
neighbor_cell.hidden_squares -= 1

# for each neighbor increase the number of mines discovered by 1
neighbor_cell.mines += 1
```

   Case 2: New cell = SAFE Cell

```
# for each neighbor reduce the number of hidden squares by 1
neighbor_cell.hidden_squares -= 1

# for each neighbor increase the number of safe cells discovered by 1
neighbor_cell.safe_cells += 1
```

c. **Update the knowledge base**

Remove the newly found cell from the all keys i.e. the equations in the knowledge base, because that cell is no longer unknown. If the cell getting removed is safe, just removing it from the knowledge base is enough. If the cell is a mine, we also need to update the value of the equation by subtracting 1 from it. For updating information about a single cell we need to iterate the entire knowledge base.

Approach followed for knowledge base update:

By baseline and by the inference algorithm, we get a list of mine cells and safe cells. We query the cell one by one and update the parameters as described in part (a) and (b) above. After the neighbor update is done, the knowledge base is getting updated.

```
mine_cell_list
safe_cell_list
for equation in the KB:
    # update of mine cells
    equation = equation - intersection(equation, mine_cell_list)
    value = value - len(intersection(equation, mine_cell_list))

    # update of safe cells
    equation = equation - intersection(equation, safe_cell_list)
```

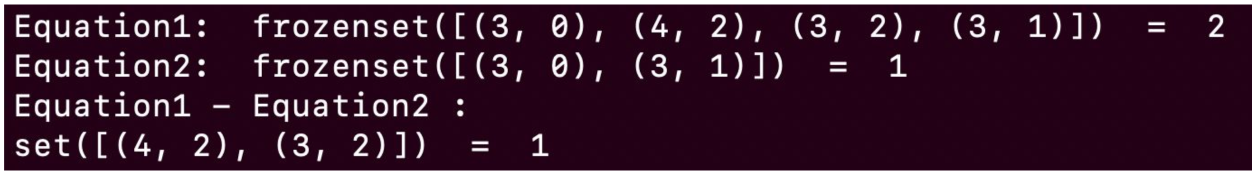
Does your program deduce everything it can from a given clue before continuing? If so, how can you be sure of this, and if not, how could you consider improving it?

Ans: Yes, our program deduces everything it can from the given clue before it continues to open a new cell randomly. For each of the cell which is opened, we run the baseline algorithm on that cell which makes sure all the local information given by that cell is fully used.

Following are the algorithms used for inference.

a. **Baseline** - As described in part 2.1
b. **Basic Inference:** Checking if the variables in the different equations are the subsets of each other.

   We are doing nothing but **subtraction of two equation**s.



```
Equation1:  frozenset([(3, 0), (4, 2), (3, 2), (3, 1)])  =  2
Equation2:  frozenset([(3, 0), (3, 1)])  =  1
Equation1 - Equation2 :
set([(4, 2), (3, 2)])  =  1
```

c. **Advanced Inference:** Checking for the intersection of two sets instead of subset. -- We can infer the mine cells from this logic

Idea: Any subset of the equation makes the equation to be a **<= equation** and when we **subtract** a **<= equation** from an **= equation**, the remaining part of the equation becomes **>=**.

Once we have the equation of the form >=, we can infer that all variables are 1 if the value of the equation is the number of variables on the LHS of equation.

This is similar to one of the bonus questions.

Equation1: Var1+Var2+...+VarN >= N implies that all variables are 1 (given the constraint that each value can be either a 0 or 1)

Pseudo Code:

```
Equation1: set1 = value1
Equation2: set2 = value2
intersection_set = set1.intersection(set2)

# if there is an intersection and the value of both the sets is different we
take the lower value. Now if removing that intersecting set makes the length
of the remaining set equation equal to the subtracted value, we can deduce
that all those cells are the mine cells
if intersection_set:
    if value1>value2:
        tmp_set = set(equation1) - intersection_set
        tmp_set_value = value1 - value2
        if len(tmp_set)>0 and len(tmp_set)==tmp_set_value:
            mine_cells.extend(list(tmp_set))

    elif value2>value1:
        tmp_set = set(equation2) - intersection_set
        tmp_set_value = value2 - value1
        if len(tmp_set)>0 and len(tmp_set)==tmp_set_value:
            mine_cells.extend(list(tmp_set))
```

Example: Understanding it with an example is much better. Suppose we have the following two equations:

Equation1: A+B+C = 2

Equation2: B+C+D = 2 => C+D <= 2

Equation3: C+D+E+F+G = 1 => C+D <= 1

Here, there are no possible reductions based on subset logic. Now, let’s consider intersection. Intersection of {B,C,D} and {C, D, E, F, G} = {C, D}

Using Equation3 we can deduce that any subset of that equation is <= 1 and based on Equation2 we can deduce that any subset of that equation is <=2.

{C, D}<=1 based on Equation3 and {C, D}<=2 based on Equation2. Combining these two, {C,D} <=1

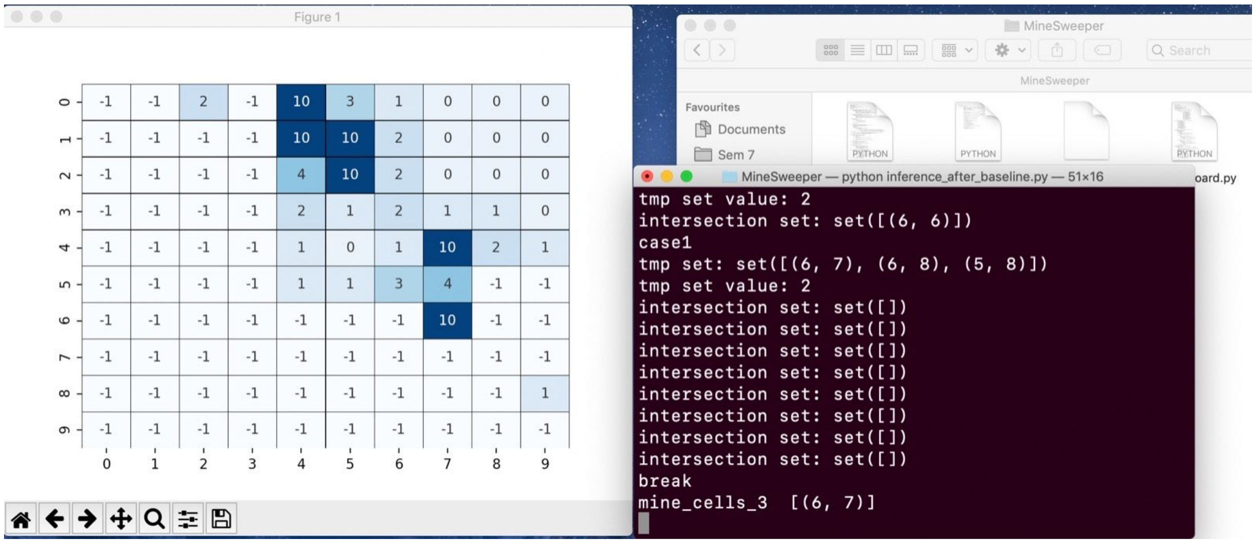
Now, updating the information in Equation2:

B+C+D=2 and C+D<=1

=> B>=1

=> B=1

A working example:



	0	1	2	3	4	5	6	7	8	9
0	-1	-1	2	-1	10	3	1	0	0	0
1	-1	-1	-1	-1	10	10	2	0	0	0
2	-1	-1	-1	-1	4	10	2	0	0	0
3	-1	-1	-1	-1	2	1	2	1	1	0
4	-1	-1	-1	-1	1	0	1	10	2	1
5	-1	-1	-1	-1	1	1	3	4	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	10	-1	-1
7	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
8	-1	-1	-1	-1	-1	-1	-1	-1	-1	1
9	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

```
tmp set value: 2
intersection set: set([(6, 6)])
case1
tmp set: set([(6, 7), (6, 8), (5, 8)])
tmp set value: 2
intersection set: set([])
intersection set: set([])
intersection set: set([])
intersection set: set([])
intersection set: set([])
intersection set: set([])
intersection set: set([])
intersection set: set([])
break
mine_cells_3  [(6, 7)]
```

Over here the algorithm was able to figure out that (6, 7) is a mine cell using the following inferences:

1. Clue from (5, 5) says that one of the [(6, 4), (6, 5), (6, 6)] is a mine
2. Clue from (5, 6) says that two of the [(6, 5), (6, 6), (6, 7)] are mines

This implies (6, 5) + (6, 6) <= 1

Since (6, 5), (6, 6), (6, 7) = 2

(6, 7) = 1

Since neither of the set is a subset of the other, the algorithm could not have been able to arrive at the above inference.

Completeness of the inference

To ensure the completeness of the inferences we are making, we continue the loop until there are no more inferences to be made. Hence, our inference is complete.

3.3 Decisions

Given a current state of the board, and a state of knowledge about the board, how does your program decide which cell to search next? Are there any risks, and how do you face them?

1. **Random choice**

   Pick a random cell from all the unexplored cells.

2. **Likelihood based random choice**

We compute the likelihood of cell being a mine based on the knowledge base and then choose the cell with the minimum likelihood. If there are multiple cells with the same likelihood, we choose one cell out of those randomly.

A cell can potentially exist in multiple places in our knowledge base. Each occurrence tells us something about the Likelihood (set_value/set_size) of the cell being a mine. Following are the approaches explored for combining the different Likelihoods obtained for the same cell:

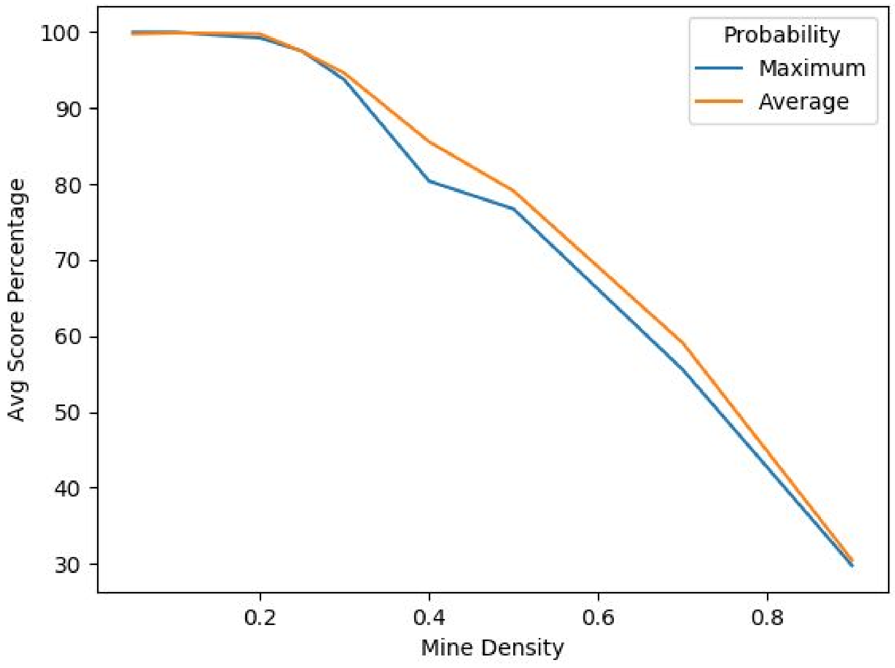
Option1: Taking max likelihood

For a given cell, we compute the max likelihood out of all the possible likelihoods. Our aim is to **avoid mine cells** while performing random select. Therefore we consider the worst case ie take the max Likelihood for each of the cells and pick the cell which has min max Likelihood.

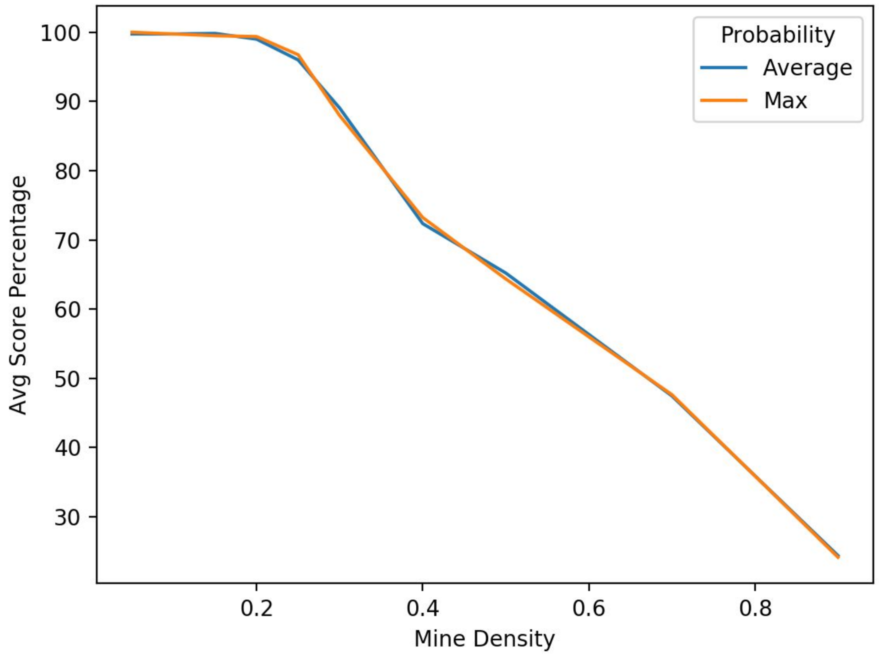
Option 2: Taking average likelihood

For a given cell, we compute the max likelihood out of all the possible likelihoods. Instead of discarding all the non-max Likelihood values, the average of all the Likelihood values could be taken. This makes sense because, if one neighbor says that cell is a mine with probability ⅛ and if other neighbor says that the cell is a mine with probability 6/8, we can give equal weight to both of these by taking an average. The next random pick could be the one having the least average Likelihood.

Following plot shows that Option 2 gives better results. This is the case when the total number of mines are known.



The below graph shows both the options perform equally well when total number of mines are not known.



Case-1: Total mines unknown

We can not say anything about the likelihood of unexplored cells whose neighbors have not been explored yet. Hence while working on the random pick, we randomly pick one cell from the set of cells whose likelihoods are unknown and the set of cells having the least likelihoods. Basically, we are trying to avoid the cells with high probability of being a mine.

Case-2: Total mines known

We know the likelihood of all the cells, thus we just pick the cells having the least likelihood. The above mentioned approach is not complete but due to computational constraints we can not apply the complete solution.

Following example explains the same (from our discussion with Professor Cowan):

1	A	2
B	C	D
E	3	F

Consider B. Given the upper left clue, the likelihood of B being a mine is 1/3. But considering the bottom center, the likelihood that B is a mine should be 3/5. However, neither of them are the real probabilities that B is a mine.

Each of the following solutions are equally likely which essentially gives us the probability of B = 0

1	X	2
		X
X	3	X

1		2
	X	X
	3	X

1		2
	X	X
X	3	

However, to conclude the same, we have to check and see how many solutions are possible. For an nXn board, it would mean verifying 2^n possibilities. Implementing the same would be computationally very expensive for large n's. Thus we followed the likelihood approach instead.

- **Performance**

1. *For a reasonably-sized board and a reasonable number of mines, include a play-by-play progression to completion or loss. Are there any points where your program makes a decision that you don't agree with?*

Ans: No, there were no points at which the program took an unacceptable decision. It always made inferences as expected or in a much better way than an average human could make.

2. *Are there any points where your program made a decision that surprised you? Why was your program able to make that decision?*

Ans: No, there were no points where our program made any surprising decision. This is because our algorithm starts by invoking the baseline algorithm first. Only when baseline makes all the possible decisions regarding the mines and safe cells do we invoke the inference method. Once the inference algorithm completes all the inferences possible, we invoke the advanced inference. The baseline is invoked again after the advanced inference to check if it can make any further decisions based upon the new and updated knowledge.

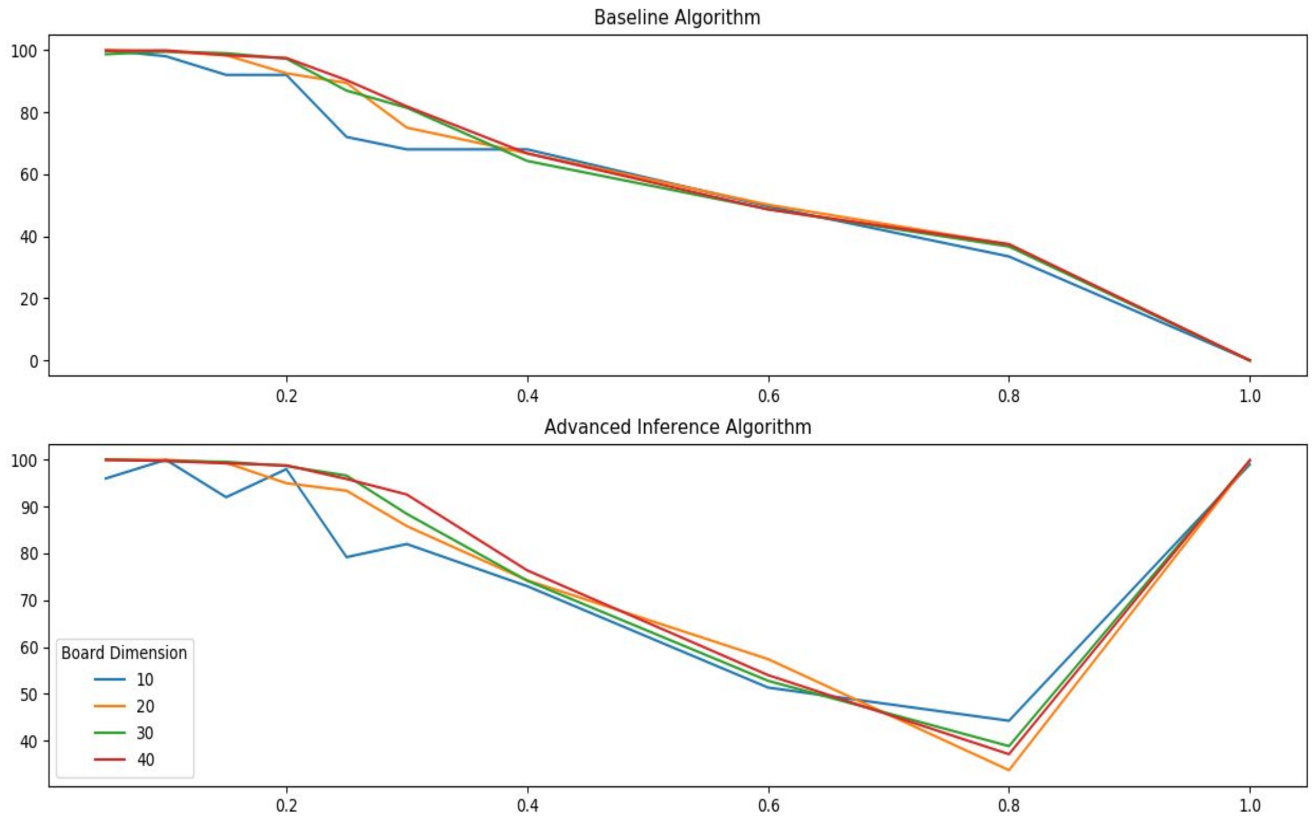
Baseline -> Basic Inference -> Advanced Inference -> Baseline …... Loop

This flow of the algorithm makes sure that the simplest and most obvious decisions are made first and these decisions update the knowledge base for the next algorithm in line to make more informed and complex inferences or decisions.

- **Performance**

1. *For a fixed, reasonable size of board, plot as a function of mine density the average final score (safely identified mines / total mines) for the simple baseline algorithm and your algorithm for comparison.*

For deeper analysis, we considered four different dimensions of 10, 20, 30 and 40 for the Minesweeper game and mine densities in the range of 0.05 to 1, where the board has the least number of mines at density 0.05 and maximum number of mines at density 1 (all the cells in the board are mines). For each dimension and each value of mine density, we solved 50 random boards by using both the Baseline and our Inference algorithms. The resulting plot is as below.

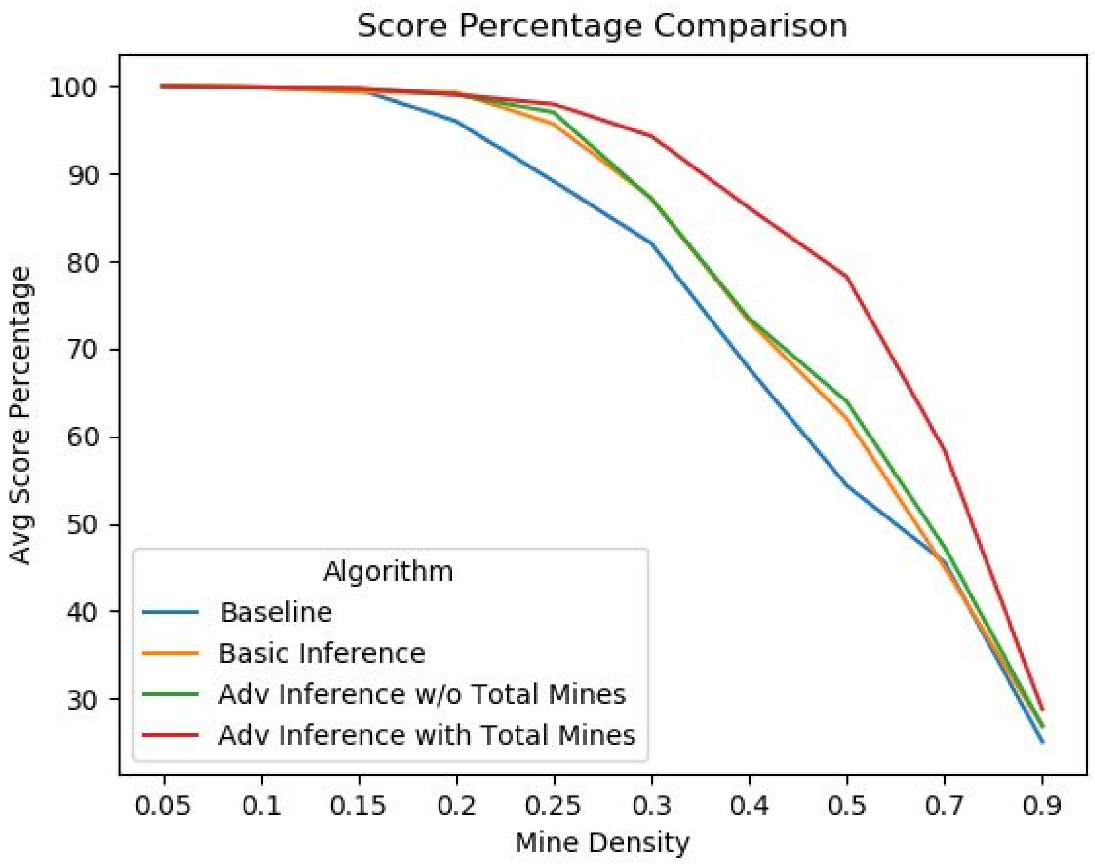


As observed, the performance of both algorithms gradually becomes independent of the dimension of the board as the mine density keeps increasing. The average score percentage keeps decreasing with increasing mine density. This is reasonable because, as the number of mines on the board increases, the probability of a randomly chosen cell being a mine increases. As a result, the average score decreases.

Special Case: Mine density = 1, that is, all the n^2 cells are mines.

1. **Baseline Algorithm**: The average score percentage of the baseline algorithm is 0 which is expected because the algorithm does not have any clues or safe cells to deduce information about the other hidden cells. So, it is not possible to figure out mine cells and the baseline always resorts to random picking of cells and all the random picks turn out to be mine cells.
2. **Advanced Inference Algorithm with Total Number of Mines**: The score percentage is around 99.8% because after the algorithm picks the first random cell which turns out to be a mine, it figures out that all the remaining cells are mines based on the information it has about the total number of mines.

Hence, by ignoring the dimension and plotting the mine density vs average score percentage by solving 50 random boards for a larger dimension = 40, we get the following graph.



The score percentage of an algorithm should be directly proportional to the amount of knowledge it possesses or computes about the clues and the mines. The graph makes sense because it proves this point.

The inferences made by Baseline < Basic Inference < Advanced Inference without Total Number of Mines < Advanced Inference with Total Number of Mines (this algorithm additionally has the information about the Total Number of Mines).

2. *When does minesweeper become 'hard'? When does your algorithm beat the simple algorithm, and when is the simple algorithm better? Why?*

   The below table shows the analysis of the score percentages of the four scenarios at mine densities 0.5 and 0.7.

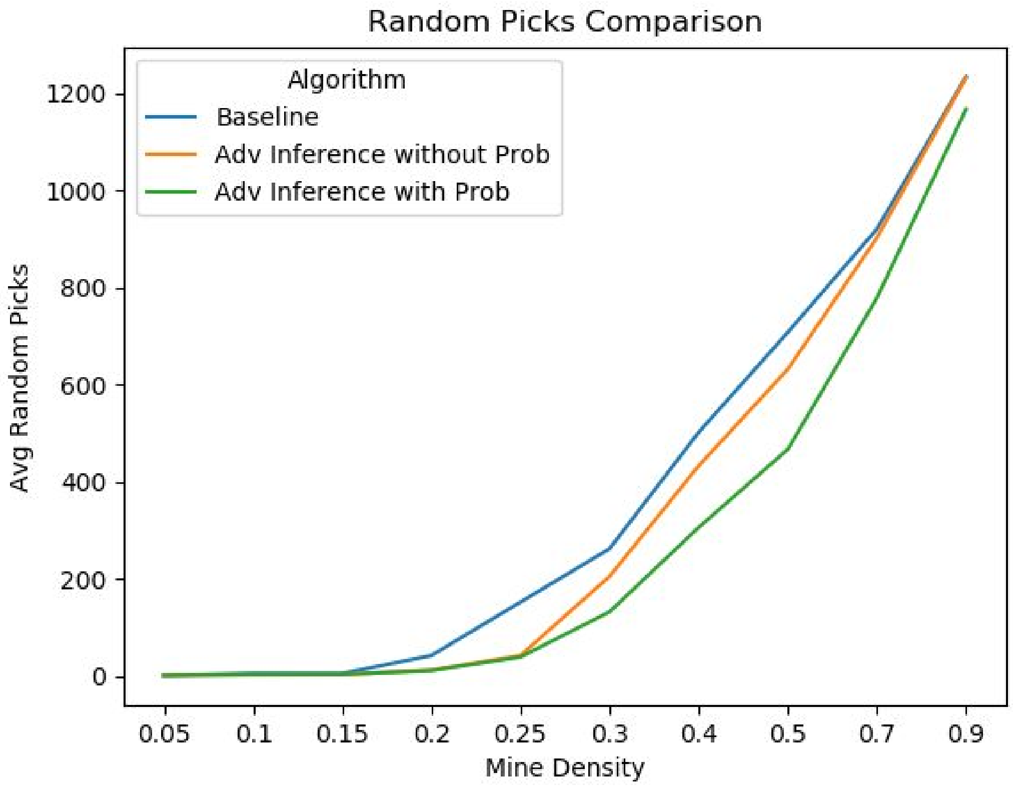
Mine Density	**Baseline**	**Basic Inference**	**Adv Inference w/o Total Mines**	**Adv Inference with Total Mines**
0.5	54.35	62	63.95	78.225
0.7	45.61	45.05	47.34	58.41

As observed from the plot data above, the minesweeper becomes hard to solve when the mine density of the board is greater than 0.5 as there is a sharp drop in the score percentage after this point and also this is when the average score percentage drops to below 50%.

We can see that the inference algorithms perform significantly better than the simple baseline algorithm in all cases. This is because:

- ➢ The rate of choosing cells randomly in the baseline algorithm is much higher than the inference algorithms as shown below. The inference algorithms pick cells only after they have exhausted all other options of deducing/inferring additional information from the knowledge base. Hence, there is a higher probability of the baseline algorithm in choosing cells which are mines.
- ➢ Also, in the inference algorithm, the addition of the probability factor in choosing random cells improves the performance as this factor estimates which cells have a higher likelihood of being a mine from the knowledge we have available and avoids choosing those cells.

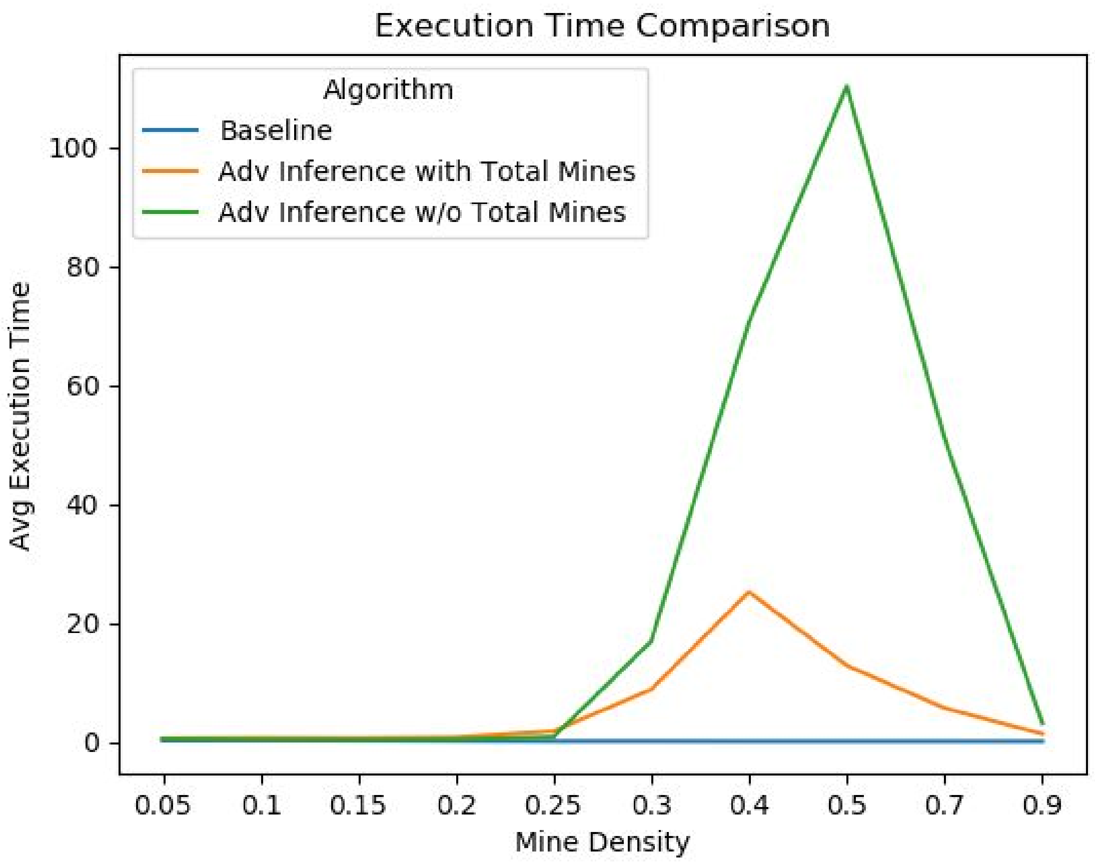
Below is the plot for the comparison of the number of random picks for each algorithm for a range of mine densities. The advanced inference performs better than baseline while the advanced inference with probability factor performs better than the one without the probability factor as it chooses random cells more carefully.



- **Efficiency**

What are some of the space or time constraints you run into in implementing this program? Are these problem specific constraints, or implementation specific constraints? In the case of implementation constraints, what could you improve on?

As shown in the graph below, the execution time of the inference algorithm is higher when compared to that of the baseline algorithm as the inference takes additional time in looking through the knowledge base at every step to check if we can deduce any more useful information regarding the covered cells. So, the higher execution time is a trade-off we have to make to achieve higher scores in the game. The inference algorithm with knowledge about the total number of mines has a lesser execution time as compared to the one without this knowledge. This is because this prior knowledge helps the algorithm in making more inferences about the mines and safe cells at every step depending on the remaining number of mines on the board and as a result, the algorithm converges faster.



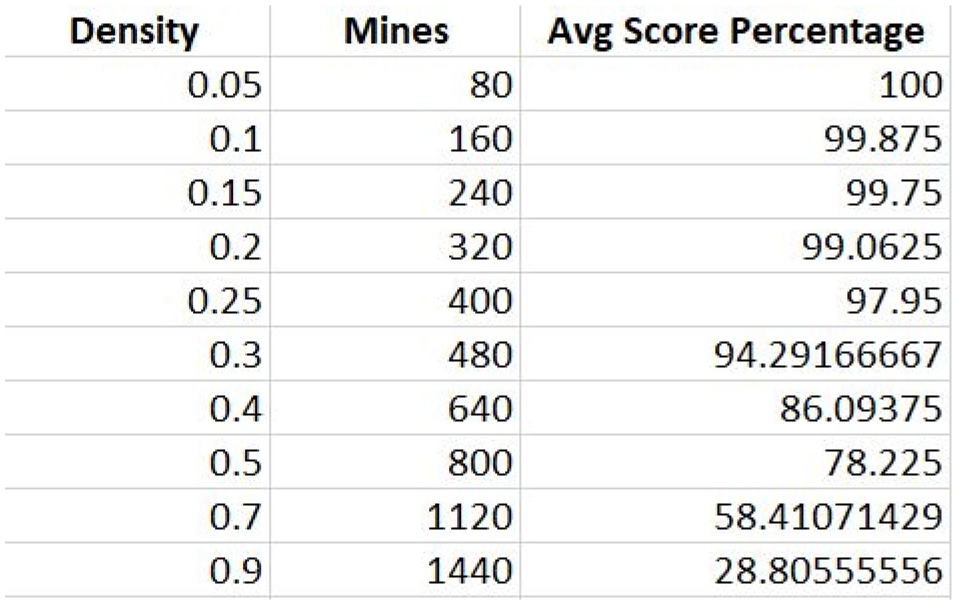
- **Improvements**

Consider augmenting your program's knowledge in the following way - tell the agent in advance how many mines there are in the environment. How can this information be modeled and included in your program, and used to inform action?

This information is added to the knowledge base in the same format the inferences are being stored, ie the mapping between the set and set value. Here set is all the cells of the board and set value is the total number of mines present.

How can you use this information to effectively improve the performance of your program, particularly in terms of the number of mines it can effectively solve? Re-generate the plot of mine density vs expected final score for your algorithm, when utilizing this extra information.

The plot for this algorithm with total number of mines is included in the plot of mine densities vs average score percentage in the previous section (Performance). The analysis data below represents the plot.



Density	Mines	Avg Score Percentage
0.05	80	100
0.1	160	99.875
0.15	240	99.75
0.2	320	99.0625
0.25	400	97.95
0.3	480	94.29166667
0.4	640	86.09375
0.5	800	78.225
0.7	1120	58.41071429
0.9	1440	28.80555556

Potential improvements:

1. Instead of the common knowledgebase, we can have cell specific knowledge and when checking for set reductions and update, we can directly update the cells that could have potentially changed.
2. Better visualization.

4 Bonus: Dealing with Uncertainty

NOTE: Final submission of this question directly to Professor Cowan

Division of Work

Every team member was equally involved in the discussions and the brainstorming that went into the implementation of the algorithms.