AI Project 2: Inference Informed Action and Minesweeper

Daniel Ying dty16 Sec: 440:06, Kyle VanWageninge kjv48 Sec: 440:05

March 23, 2021

Submitter:

Honor Code:

I abide to the rules laid in the Project 2: Minesweeper description and I have not used anyone else's work for the project, and my work is only my own and my group's.

I acknowledge and accept the Honor Code and rules of Project 2.

Signed: Daniel Ying dty16, Kyle VanWageninge kjv48

Workload:

Daniel Ying: Formated the report, wrote the report in latex code. Compiled the results and answers onto the report.

Kyle VanWageninge: Coded basic and improved agent as well as implemented discussed better bonus. Implemented code to test each algorithm and recorded the results.

Together: Discussed how we would implement our improved algorithm to run and designed the pseudo code and thoughts about our inference. Went over each question together and answered the problems together

Problem 1

Representation: How did you represent the board in your program, and how did you represent the information / knowledge that clue cells reveal? How could you represent inferred relationships between cells?

ANSWER:

ACRONYM: Our advanced algorithm is called Mine Elimination and Avoidance Network, aka M.E.A.N.

Description:

To represent the board and the information/knowledge revealed by the clue cells, we set up two different boards: a 'hint' board and a 'game' board. The hint board would be holding the inferred relationships between the cells. While the game board represents a square board with a given dimensions that contains '?' in its grids (all are setup with (x,y) double array). The game board would also hold the final revealed results of the board.

After the board is set and the mines are randomly placed in the board, our code would use the hint board to identify the relationships of each grid with its neighbors and setting up to represent the inferred relationships between the grids. With our advanced sweeper (the improved algorithm), we would run query a grid in hint board, we would identify it as a clue (a number representing the bomb(s) next to it). When the clue we found in the hint board fulfilled our relationship statement (as in if it leads to any inferences in our improved sweeper algorithm), then the clue would create an equation in which we store in our knowledge base (further information about the equation and knowledge base would be explained in the next question).

From the inference in the knowledge base, we could then use the (x,y) position of the hint board and infer whether the block would be marked with a 'c' (meaning this cell is cleared to be queried) or 'M' (meaning the cell is a mine that need to be not queried and should be avoided) in the same position (x,y) on the game board.

Problem 2

Inference: When you collect a new clue, how do you model / process / compute the information you gain from it? In other words, how do you update your current state of knowledge based on that clue? 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?

ANSWER:

Stated previously, the clues we obtained from the hint board's grid would be stored as a series of equations (depending on the number of neighbors the grid has).

For example:

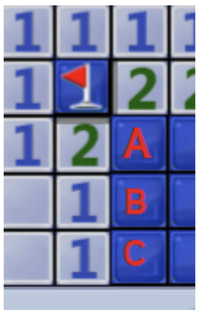

Figure 1: An example scenario for minesweeper Equation that forms three equations based on A, B, and C.

$$A + B = 1$$

$$A + B + C = 1$$

$$B + C = 1$$

Knowledge Base format example:

(x,y,0 constraint,-1); [-1 means the constraint is not final, if 0 or 1 then constraint is final]

Another subset of equations (each variable is either a 1 or a 0, as in is it a bomb or not):

$$A + B + C = 1$$

$$D + F = 0$$

$$A + C = 1$$

Let's say the position of the clue for the three equations above is (0,1) [not Figure 1], then our knowledge base would then store a set of equations as:

[(1,0,0,-1), (0,0,0,-1),(1,1,0,-1), 2]

The statement means when positions (0,0), (0,0), (1,1) are added together, they would equal to 2 mines. This is a set of equations held in knowledge base.

Now if a clear cell 'c' is added, the improved algorithm would look through the knowledge base to find the equations with the position (x,y) for they would be marked thus no longer needed. So lets say the position (1,0) has been cleared, thus we would remove its equation from the knowledge base.

Now the knowledge base would look like:

[(0,0,0,-1),(1,1,0,-1),2]

If a 'M' (mine flag) is added, the improved algorithm would then search through the knowledge base for any equation that has the position (x,y) and remove that equation from the series of equations. Then the algorithm would minus the total number of mines by 1, because the mine position was found. Thus, for example, let's say the previously mentioned (1,0) was not a clear, but a mine; we would instead update the knowledge base as:

[(0,0,0,-1),(1,1,0,-1),1]

After each clue (equation) was added to the knowledge base, the algorithm would then try to find basic deductions, for example:

$$A + B + C = 0$$

This would mean positions A, B, C are all clear to be queried.

$$A + B + C = 3$$

This would mean positions A, B, C are all bombs.

A clue would be added to the knowledge base one at a time, to make sure all basic deductions are made and none would be missed. After the basic deductions are completed and all clear grids that can be queried by the basic deduction, the algorithm would then create subsets of equations.

The use of basic deduction allows us to save time instead of creating a subset of equations each time a new clue is found. This is especially useful since our algorithm's subset checking process takes a bit of time to complete. Then after the subsets are created, the algorithm would then double check to see if any more deductions can be made by the basic deductions in the newly updated knowledge base.

Only if no more basic deductions can be found, would our algorithm move forward to implement backtracking. The backtracking would combine the equations (that have at least one variable in common) in the subsets together as one compiled equation, thus adding up their clues' total mines as well. This would then produce a list of compiled equations and each variable would be a constraint in that solution.

At this point, if a given position (x,y) is clear, it would be greater than 85% in all the given compiled equations.

We will explain subset and backtracking algorithm in more detail.

SUBSETS:

To begin the subset step, we first take the first equation in the knowledge base and send it to the isSub() function which checks if the equation is a subset of any of the other equations in the knowledge base. isSub() does this by taking the equation to pass it into the function and iterate through the the knowledge base to search for equations that are:

- 1) bigger than the equation
- 2) has all the variables from the inputted equation inside the one being iterated now.

Once an equation that fulfills these conditions is found, we would pop those variables off of the equation in the knowledge base and update that equation without the variables.

After the entire knowledge base has updated all the subsets that fulfills the conditions mentioned above, we will run the basic inference to check if after updating the subsets would reveal obvious relationships between the grids for our basic deduction algorithm to process.

In most cases, if a subset could be found, then it would usually reveal a relationship between the grids for the basic deduction algorithm to detect and process. If not, then the algorithm would move forward to next step: backtracking.

Example of Subset

- Knowledge base: (Run subsets)
- A + B + C = 1 -> [(1, 0, 0, -1), (0, 0, 0 ,-1), (1, 1, 0, -1), 1]
- A + B = 1 -> [(1, 0, 0, -1), (0, 0, 0 ,-1), 1]
- C + D = 1 -> [(1, 1, 0, -1), (2, 1, 0, -1),1]
- Knowledge base: (After completing a subset run, run basic deduction)
- C = 0 -> [(1, 1, 0, 0), 0]
- A + B = 1 -> [(1, 0, 0, -1), (0, 0, 0 ,-1), 1]
- D = 1 -> (2, 1, 1, 1),1]
- Knowledge base: (Final knowledge base after subset algorithm runs)
- A + B = 1 -> [(1, 0, 0, -1), (0, 0, 0 ,-1), 1]

BACKTRACKING:

For our backtracking algorithm, the first step is to pick the first equation in the knowledge base to iterate. Then we take the variables in the equation and add them to another list (called unique variables), which would be used to check later. Then we search every equation in the knowledge base and match any equations that include one or more of the unique variables list and compile those equations together (add their clue at the end).

Once the compiled equation is complete, we will check if all the unique variables in the list are now assigned to different constraints. This is accomplished by using a loop of 2(̂number of unique variables) and convert each iteration of the loop to a binary number, create a list of that binary number, and assign each binary[i] to the corresponding uniqueVariable[i]. This allows us to try all possible unique variable combination constraints. Then the combination constraints that is a solution would be saved to another list.

Here is where we will search through the solutions and try to find an (x,y) position that was a clear cell with a success probability of greater than 85% (grid is clear and not a mine) in the solutions. This single (x,y) position would then be sent back to the algorithm to be queried in the next step.

Example of backtracking:

- Knowledge base: (Run backtracking step 1, creating equations)
- A + B + C = 1 => [(1, 0, 0, -1), (0, 0, 0 ,-1), (1, 1, 0, -1), 1]
- A + D + C = 1 => [(1, 0, 0, -1), (3, 0, 0 ,-1), (1, 1, 0, -1), 1]
- C + F = 1 => [(1, 1, 0, -1), (3, 1, 0, -1),1]
- (Create compiled equation)
- Unique variables: A, B, C, D, F
- A + B + C + A + D + C + C + F = 3
- (Find solutions that fit the compiled equation)
- 1) A = 1, B = 1, C = 0, D = 0, F = 0 -> 3
- 2) A = 1, B = 0, C = 0, D = 0, F = 1 -> 3
- 3) A = 1, B = 0, C = 0, D = 1, F = 0 -> 3
- 4) A = 0, B = 0, C = 1, D = 0, F = 0 -> 3
- 5) A = 0, B = 1, C = 0, D = 1, F = 1 -> 3

  C = 0 is the solution in 4 out of 5 of the equations, thus C has an 80% chance of being clear. Now in our algorithm, if any variable in this equation get over 85% of being clear, then out of all of the possible solutions, we would like to query this (x,y) grid and send it to the knowledge base

- Knowledge base: (Say C is greater than 85% and we queried it and it was a clear)
- A + B = 1 -> [(1, 0, 0, -1), (0, 0, 0 ,-1), 1]
- A + D = 1 -> [(1, 0, 0, -1), (3, 0, 0 ,-1), 1]
- F = 1 -> [(3, 1, 1, 1),1]

  So, by using backtracking, we found a clear grid as well as a mine (0 is clear, 1 is mine).

Problem 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?

ANSWER:

If we are given a current state of the board, and a state of knowledge base about the board, our algorithm would decide the next cell to search by a series of steps:

0. Pick a random grid on the board.

1. Infer between the grid and its neighbors.

2. Search through the board for any grids marked 'c' as in clear.

3. Search for any basic deduction that can be applied on the clear grids.

4. If no more clear grids can be implemented by basic deduction, create subsets with equation in knowledge base.

5. Once again, infer between the grids.

6. Search for any grids marked clear.

7. Search for any basic deduction that can be applied on the clear grids.

8. If no more clear grids found and no more subsets can be created from knowledge base, use backtracking with the equations in knowledge base.

9. Search for grids marked clear.

10. Search for any basic deduction that can be applied on the clear grids.

11. If no more grids are marked clear, pick a random hidden grid.

12. Repeat steps 1-11 until the board is complete.

Problem 4

Performance: 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? Are there any points where your program made a decision that surprised you? Why was your program able to make that decision?

ANSWER:

A step by step process for a 7x7 board with 13 mines.

Start:

```
? ? ? ? ? ? ?
? ? ? ? ? ? ?
? ? ? ? ? ? ?
? ? ? ? ? ? ?
? ? ? ? ? ? ?
? ? ? ? ? ? ?
? ? ? ? ? ? ?
```

Random grid chosen:

```
? ? ? ? ? ? ?    ? ? ? ? ? ? ?    ? ? ? ? ? ? ?
? ? ? ? ? ? ?    ? ? ? ? ? ? 1    ? ? ? ? ? 1 1
? ? ? ? ? ? ?    ? 2 ? ? ? ? ?    ? 1 ? ? ? ? ?
? ? ? ? ? ? ? →  ? ? ? ? ? ? ? →  ? ? ? ? ? ? ?
? ? ? ? ? ? ?    ? ? ? ? ? ? ?    ? ? ? ? ? ? ?
? ? ? ? ? ? ?    ? ? ? ? ? ? ?    ? ? ? ? ? ? ?
? ? ? ? ? ? ?    ? ? ? ? ? ? ?    ? ? ? ? ? ? ?
```

Using subset, 3 cells are cleared ('c'):

```
? ? ? ? c ? ?
? ? ? ? c 1 1
? 1 ? ? c ? ?
? ? ? ? ? ? ?
? ? ? ? ? ? ?
? ? ? ? ? ? ?
? ? ? ? ? ? ?
```

Choose a clear cell to query in each step (query 4 times):

$$\begin{array}{ccccccc} ? & ? & ? & c & 0 & c & ? \\ ? & ? & ? & c & c & 1 & 1 \\ ? & 1 & ? & ? & c & ? & ? \\ ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? & ? \end{array} \rightarrow \begin{array}{ccccccc} ? & ? & 1 & 0 & 0 & 1 & ? \\ ? & ? & 1 & c & c & 1 & 1 \\ ? & 1 & ? & ? & c & ? & ? \\ ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? & ? \end{array}$$

Using basic deduction we conclude a mine (M) position: (still querying clear grid one at a time)

$$\begin{array}{ccccccc} ? & ? & 1 & 0 & 0 & 1 & M \\ ? & ? & 1 & 0 & c & 1 & 1 \\ ? & 1 & c & c & c & c & c \\ ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? & ? \end{array}$$

Query 4 times (4 clear grids):

$$\begin{array}{ccccccc} ? & ? & 1 & 0 & 0 & 1 & M \\ ? & ? & 1 & 0 & c & 1 & 1 \\ ? & 1 & 1 & 1 & c & c & 0 \\ ? & ? & ? & ? & ? & c & 1 \\ ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? & ? \end{array}$$

---

next 2 queries:

$$\begin{array}{ccccccc} ? & ? & 1 & 0 & 0 & 1 & M \\ ? & ? & 1 & 0 & c & 1 & 1 \\ ? & 1 & 1 & 1 & 1 & 0 & 0 \\ ? & ? & ? & ? & 2 & c & 1 \\ ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? & ? \end{array}$$

The last c in the previous step is queried using the blue grid's information, allowing us to deduce the next mine's position and 3 clear grids:

$$\begin{array}{ccccccc} ? & ? & 1 & 0 & 0 & 1 & M \\ ? & c & 1 & 0 & 0 & 1 & 1 \\ ? & 1 & 1 & 1 & \color{blue}{1} & 0 & 0 \\ ? & c & c & M & 2 & 1 & 1 \\ ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? & ? \end{array}$$

Query clear grid at (1,1), another mine is found:

$$\begin{array}{ccccccc}
? & M & 1 & 0 & 0 & 1 & M \\
? & 2 & 1 & 0 & 0 & 1 & 1 \\
? & 1 & 1 & 1 & 1 & 0 & 0 \\
? & c & c & M & 2 & 1 & 1 \\
? & ? & ? & ? & ? & ? & ? \\
? & ? & ? & ? & ? & ? & ? \\
? & ? & ? & ? & ? & ? & ?
\end{array}
\longrightarrow
\begin{array}{ccccccc}
? & M & 1 & 0 & 0 & 1 & M \\
? & 2 & 1 & 0 & 0 & 1 & 1 \\
? & 1 & 1 & 1 & 1 & 0 & 0 \\
? & 1 & 1 & M & 2 & 1 & 1 \\
? & c & c & c & ? & ? & ? \\
? & ? & ? & ? & ? & ? & ? \\
? & ? & ? & ? & ? & ? & ?
\end{array}$$

Query next 3 clear grids (no more basic deduction can be made, so we create subsets for our knowledge base):

$$\begin{array}{ccccccc}
? & M & 1 & 0 & 0 & 1 & M \\
? & 2 & 1 & 0 & 0 & 1 & 1 \\
? & 1 & 1 & 1 & 1 & 0 & 0 \\
? & 1 & 1 & M & 2 & 1 & 1 \\
? & 2 & 3 & 2 & ? & ? & ? \\
? & ? & ? & ? & ? & ? & ? \\
? & ? & ? & ? & ? & ? & ?
\end{array}$$

This clears 2 more grids and deduce another mine position. The subset helped because the blue 1 only have two hidden grids and one had to be a mine, while the clues next to it also had one undiscovered mine. If this was only the basic algorithm, it would not have been able to detect this mine at this step.

$$\begin{array}{ccccccc}
? & M & 1 & 0 & 0 & 1 & M \\
? & 2 & 1 & 0 & 0 & 1 & 1 \\
? & 1 & 1 & 1 & 1 & 0 & 0 \\
? & 1 & 1 & M & 2 & 1 & 1 \\
? & 2 & 3 & 2 & C & M & C \\
? & ? & ? & ? & ? & ? & ? \\
? & ? & ? & ? & ? & ? & ?
\end{array}$$

Query 2 new clear grids, and no more basic deduction can be made, thus we create some more subsets. After running the subsets, but none can be found, thus we try backtracking instead. But no variables in knowledge base that is greater than 85% can be found. Thus we pick a random grid to query instead:

$$\begin{array}{ccccccc}
? & M & 1 & 0 & 0 & 1 & M \\
? & 2 & 1 & 0 & 0 & 1 & 1 \\
? & 1 & 1 & 1 & 1 & 0 & 0 \\
? & 1 & 1 & M & 2 & 1 & 1 \\
? & 2 & 3 & 2 & 4 & M & 2 \\
? & ? & ? & ? & ? & ? & ? \\
? & ? & ? & ? & ? & ? & ?
\end{array}$$

The random grid reveals a mine position but does not give us any updates on the current equations in the knowledge base. So find another random position again:

?	*M*	1	0	0	1	*M*
?	2	1	0	0	1	1
?	1	1	1	1	0	0
?	1	1	*M*	2	1	1
?	2	3	2	4	*M*	2
?	?	?	?	?	?	?
?	?	?	?	?	?	*M*

Now with the new 2 at the bottom of the board, some subsets could be made but nothing that leads to a clear grid. So we ran backtracking but again nothing can be cleared. A new random grod is queried again:

?	*M*	1	0	0	1	*M*
?	2	1	0	0	1	1
?	1	1	1	1	0	0
?	1	1	*M*	2	1	1
?	2	3	2	4	*M*	2
?	?	?	?	?	?	?
?	?	?	2	?	?	*M*

After this clue, we can create some subsets that lead to a new clear grid:

c	*M*	1	0	0	1	*M*
?	2	1	0	0	1	1
?	1	1	1	1	0	0
1	1	1	*M*	2	1	1
?	2	3	2	4	*M*	2
?	?	?	?	?	?	?
?	?	?	2	?	?	*M*

After querying we found another clear grid that will lead to another mine being found:

1	*M*	1	0	0	1	*M*
2	2	1	0	0	1	1
M	1	1	1	1	0	0
1	1	1	*M*	2	1	1
?	2	3	2	4	*M*	2
?	?	?	?	?	?	?
?	?	?	2	?	?	*M*

With this new position, another clear grid is found:

1	*M*	1	0	0	1	*M*
2	2	1	0	0	1	1
M	1	1	1	1	0	0
1	1	1	*M*	2	1	1
c	2	3	2	4	*M*	2
?	?	?	?	?	?	?
?	?	?	2	?	?	*M*

This new queried cell gives us 2 new mine positions as well as another clear grid to query:

$$\begin{array}{ccccccc}
1 & M & 1 & 0 & 0 & 1 & M \\
2 & 2 & 1 & 0 & 0 & 1 & 1 \\
M & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & M & 2 & 1 & 1 \\
2 & 2 & 3 & 2 & 4 & M & 2 \\
M & M & c & ? & ? & ? & ? \\
? & ? & ? & 2 & ? & ? & M
\end{array}$$

Another mine and clear is found:

$$\begin{array}{ccccccc}
1 & M & 1 & 0 & 0 & 1 & M \\
2 & 2 & 1 & 0 & 0 & 1 & 1 \\
M & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & M & 2 & 1 & 1 \\
2 & 2 & 3 & 2 & 4 & M & 2 \\
M & M & c & M & c & ? & ? \\
? & ? & ? & 2 & ? & ? & M
\end{array}$$

Again, another mine and clear grid is found:

$$\begin{array}{ccccccc}
1 & M & 1 & 0 & 0 & 1 & M \\
2 & 2 & 1 & 0 & 0 & 1 & 1 \\
M & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & M & 2 & 1 & 1 \\
2 & 2 & 3 & 2 & 4 & M & 2 \\
M & M & 3 & M & 4 & M & c \\
? & ? & ? & 2 & ? & ? & M
\end{array}$$

Reveal another clear cell, then the rest would be finished by basic deduction:

$$\begin{array}{ccccccc}
1 & M & 1 & 0 & 0 & 1 & M \\
2 & 2 & 1 & 0 & 0 & 1 & 1 \\
M & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & M & 2 & 1 & 1 \\
2 & 2 & 3 & 2 & 4 & M & 2 \\
M & M & 3 & M & 4 & M & 3 \\
? & ? & ? & 2 & ? & ? & M
\end{array}
\longrightarrow
\begin{array}{ccccccc}
1 & M & 1 & 0 & 0 & 1 & M \\
2 & 2 & 1 & 0 & 0 & 1 & 1 \\
M & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & M & 2 & 1 & 1 \\
2 & 2 & 3 & 2 & 4 & M & 2 \\
M & M & 3 & M & 4 & M & 3 \\
? & ? & c & 2 & M & 3 & M
\end{array}
\longrightarrow
\begin{array}{ccccccc}
1 & M & 1 & 0 & 0 & 1 & M \\
2 & 2 & 1 & 0 & 0 & 1 & 1 \\
M & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & M & 2 & 1 & 1 \\
2 & 2 & 3 & 2 & 4 & M & 2 \\
M & M & 3 & M & 4 & M & 3 \\
? & M & 3 & 2 & M & 3 & M
\end{array}
\longrightarrow$$

$$\begin{array}{ccccccc}
1 & M & 1 & 0 & 0 & 1 & M \\
2 & 2 & 1 & 0 & 0 & 1 & 1 \\
M & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & M & 2 & 1 & 1 \\
2 & 2 & 3 & 2 & 4 & M & 2 \\
M & M & 3 & M & 4 & M & 3 \\
3 & M & 3 & 2 & M & 3 & M
\end{array}$$

After running both basic deductions and creating subsets, we were able to obtain a 92% success rate and avoided 12 mines (1 mine was exploded). Though backtracking was more useful on bigger boards but it is harder to iterate step by step on these large boards.

Result:

- Mines exploded: 1
- Mines avoided: 12
- Success rate: 0.92

Unexpected parts:

During the play by play process, we did not find any moves that our algorithm made we would disagree with. There was one move during the play by play that surprised us. It was when the board took two clues (both equals to 1) and produced three clear grids (shown below).

?	?	?	?	*c*	?	?
?	?	?	?	*c*	1	1
?	1	?	?	*c*	?	?
?	?	?	?	?	?	?
?	?	?	?	?	?	?
?	?	?	?	?	?	?
?	?	?	?	?	?	?

If a human was playing, it would take them a while to see that those 3 grids have to be clear because the second number clue (equals to 1) was found. The algorithm was able to make the fast determination by using the subset algorithm. When the second clue was added, the algorithm used subsets to deduce that there are 3 clear grids.

Problem 5

Performance: 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. This will require solving multiple random boards at a given density of mines to get good average score results. Does the graph make sense / agree with your intuition? When does minesweeper become 'hard'? When does your algorithm beat the simple algorithm, and when is the simple algorithm better? Why? How frequently is your algorithm able to work out things that the basic agent cannot?

ANSWER:

We ran the program with a board size dimension of 25x25 and the density of the mines started from 10% during the first run, then slowly increase the density by 10% for each run after the first run; until we reach 90%. 100 new boards were created and tested for each density and compiled by averaging each run.

Basic and Advanced Algorithm Success Rate for 10 to 90% Mine density:

Comparison of Basic and Adv. Algor.									
Algorithm	10%	20%	30%	40%	50%	60%	70%	80%	90%
Basic	0.99	0.93	0.79	0.64	0.55	0.49	0.43	0.37	0.26
Adv.	0.99	0.98	0.86	0.71	0.60	0.51	0.45	0.37	0.26

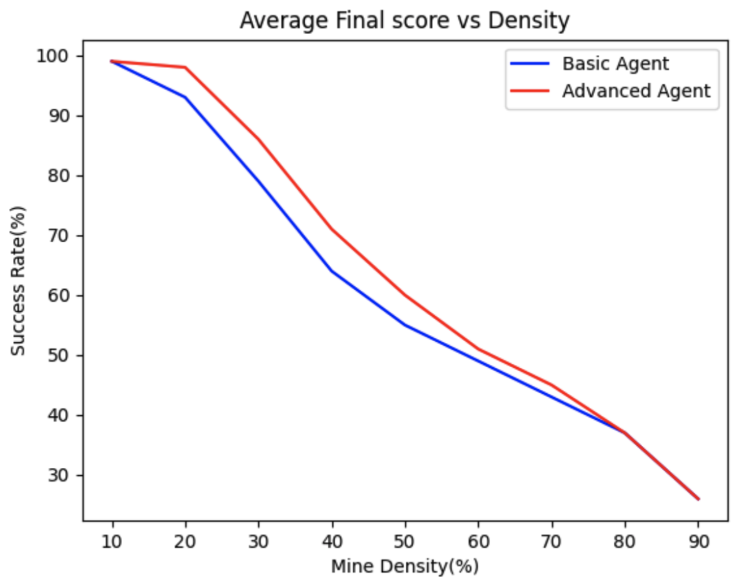

Figure 2: The comparison between average final score of the basic and advanced algorithm along the mine density (x-axis).

The graph above showed that our expectation of how the algorithm should run was accurate with the actual results. When the mine density was about 10 to 20%, the results of the basic and advanced algorithms are also very similar to one another in success rate. This observation was because more of the time the algorithm was running, it is hitting clues that equals 0, meaning that most of the board is solvable by just using the basic algorithm and the advanced algorithm was not needed to be implemented. Since the basic algorithm is only doing basic inferences, it gets inhibited when no more cells can be deduced as clear.

Thus, our advanced algorithm has two extra steps. When the basic agent would start to randomly pick grids in hopes of getting enough information to restart the basic inference again, the advanced algorithm's two extra steps would involve the subset and backtracking thus identifying the clear grid when the basic algorithm cannot.

Referring back to the graph, when the board starts to get 'harder' around 30% mine density, the minesweeper would exhibit a big drop in success rate. The rate would drop from mid to high 90s and decrease to 80%'s and below. This continues to drops as the mine density increases, showing the algorithm's difficulty to get successful results increase as the mine density increases. In terms of the success rate, the advanced algorithm produced better results than the basic algorithm, although the basic algorithm had the faster run-time. So when picking which algorithm to run, one would need to decide which is more important: success rate or run-time.

The advanced algorithm had the higher success rate because it is creating subsets to be stored in the knowledge base and using backtracking to clear cells which would otherwise be unclearable with the basic algorithm. The success of the advanced algorithm was also the main cause of the longer run-time than that of the basic algorithm, because the advanced algorithm is running a more complicated process, while the basic algorithm is just picking grids at random.

Looking at a board (which is 25x25 with 30% density: 187 mines), the advanced algorithm was able to avoid 165 mines while the basic algorithm could only avoid 152 mines. This shows the advanced algorithm was able to use it's more meticulous algorithm to clear 13 more grids which the basic algorithm would miss, thus resulting in a 5% higher success rate in the advanced algorithm. The advanced algorithm's 5% to 7% higher success rate was maintained for mine densities over 30%, showing that the advanced algorithm is frequently more capable in identifying grids in conditions when the basic algorithm cannot. However, when the mine density reaches to 70% and higher, the success rate difference between the basic algorithm and the advanced algorithm is relatively close due to the amount of mines placed on the board.

Problem 6

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?

ANSWER:

For our algorithm's time constraint, we were able to maintain it at a relatively low level. This was accomplished by the application of basic deduction and improved algorithm in our program. By implementing basic deduction in obvious situations (like A+B+C=3 would be A,B,C each equal 1), we were able to prevent the unnecessary time constraint if we used our improved algorithm instead, which in this case would involve searching through the knowledge base, inferring the clues, and finding the position (x,y) equation (making it tedious compared to the straight forward implementation of basic deduction).

The time constraint difference between the basic deduction and our improved algorithm is caused by our subset algorithm (taking the largest chunk of time). Thus, by using the basic deduction until no more deductions can be made, we were able to greatly decrease our run-time and maintain a rather efficient time constraint.

While for our backtracking algorithm, once it calculates a (x,y) position's grid to have a possibility of at least 85% of being a clear grid, the algorithm would exit without running the other equations. This newly calculated (x,y) grid would be used to query and give enough information to begin a new round of basic deductions again, thus saving time. But this tactic became not as effective as before once the dimensions of the board is set over 40x40. Once the dimensions reaches over 40x40, it would take almost 3 minutes to run at a 30% mine density.

While for a 25x25 board with a mine density of greater than 60%, the advanced algorithm would take 1 to 2 minutes to produce the results. The basic algorithm on the other hand would run at almost half of advanced algorithm's time.

The main cause of the long run-time is due to the problem constraints and our improved algorithm's own implementation constraints. With our algorithm's constraints, the problem arises when a lot of inferences are being made when the for-loops are checking the entire knowledge base to reach the final conclusion for each cell.

Problem 7

BONUS 2: Better Decisions.

ANSWER:

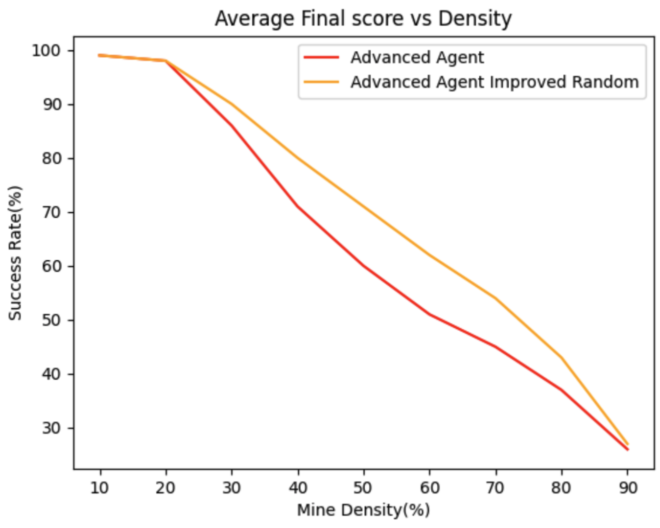

Figure 3: Comparison of the performance of advanced algorithm with improved random and without.

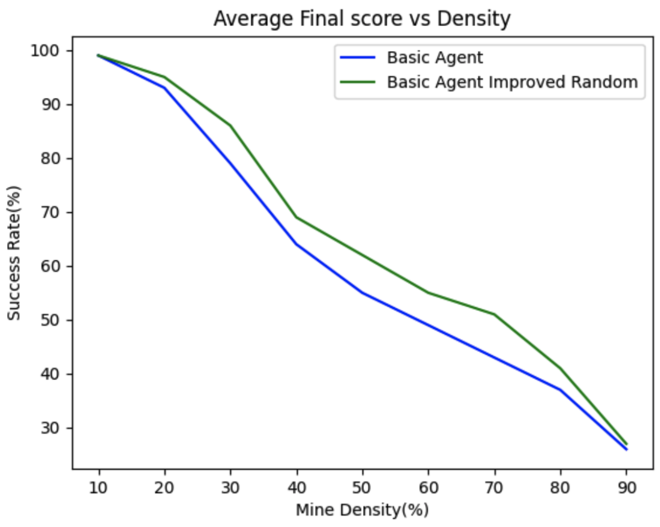

Figure 4: Comparison of the performance of basic algorithm with improved random and without.

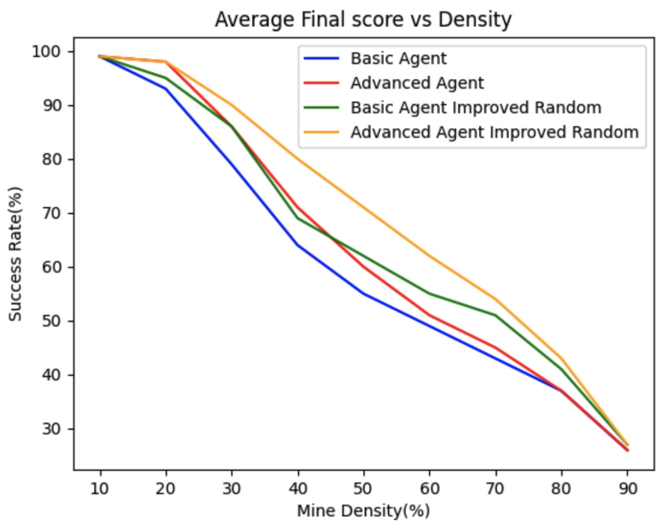

Figure 5: Comparison of the final success rate of the algorithms with improved random and without.

The table that compares the success rate results of the 4 algorithms:

2 algorithms with improved rand and 2 without									
Algorithm	10%	20%	30%	40%	50%	60%	70%	80%	90%
Basic	0.99	0.93	0.79	0.64	0.55	0.49	0.43	0.37	0.26
Basic with	0.99	0.95	0.86	0.69	0.62	0.55	0.51	0.41	0.27
Adv.	0.99	0.98	0.86	0.71	0.60	0.51	0.45	0.37	0.26
Adv. with	0.99	0.98	0.90	0.80	0.71	0.62	0.54	0.44	0.27

We ran the algorithms at a 25x25 board with a mine density that starts from 10%, increasing by 10% during each run until it reaches 90%. We created 20 new boards to test the algorithms and compiled each algorithm's score for each mine density. Unlike the previous performance section (which we tested to 100 boards), we did not do 100 boards because the run-time would be too high, thus we ran the test (20 new boards) twice and got similar results for both tests (difference of 0.004).

To improve our algorithm's random selection process of hidden grids, we would have the algorithm pick the hidden grid with the least number of total clues that surround the grid. Next, we would pick the grids that were:

- 1) Closer to the other information, thus this grid would provide more information for the algorithm to work on
- 2) Since the algorithm chooses the grid with the least total number of clues, it would then pick a cell that has the highest chance of being cleared.

These graphs showed us that because of a better random grid selection process, the algorithm was able to obtain better information of the grid's relationship with its neighbors. Looking at the basic algorithm with the improved random selection, the percentages only went up by around 5 to 6% if we compare it with the basic algorithm without the improved random selection. We expected this, because in the basic algorithm, it will have to either get all mines or all clears. Thus with random picking, it means that neither of these choices could be made, so 1 or more random grids would have to chosen instead. And with cells closer to the rest of the information, the lower the chances are of it being a mine, which would increase the success rate of avoiding a mine by a bit.

While for our advanced algorithm, it increase almost by 7 to 9% (compared to advanced without the improved random selection) when we applied the improved random selection process to it. We believe this is because the advanced algorithm can deduce more information than basic algorithm by also implementing the subsets in the knowledge base and inferring between the multiple clues (especially those close to each other). This allows the board to avoid getting information of the top left section of the board when it is actually randomly picked a grid in the bottom right. If the situation mentioned in the previous sentence did happen, the algorithm would be unable to deduce any grids from connecting bottom right grid to the information of top left section of the board, wasting run-time. Thus it would be more efficient for the algorithm to make another random grid selection instead, thus eliminating the possibility of the mentioned situation to happen.