Minesweeper

Contributors :

Bhavya Tiwari - bt290
Himanshu Jain - hj288
Nidhi Arun Harwani - nh417
Shishir Umesh - su90

3) Questions and Writeup

Representation :

The board is represented as a 2D array of dimensions entered by the user. The information revealed by any cell has been represented as the binary combinations of its neighboring cells, where 0 = No Mine and 1 = Mine

For Example :

	a	b
	c	s(3)
	d	e

The cell s reveals a number 3. This means there are 3 mines and 2 clear cells among its neighbors a,b,c,d and e. We still don't know which ones exactly based on just this information. All we can conclude are the possible combinations ($5C3$ in this case) that can arise out of this piece of information, which is represented as follows :

a	b	c	d	e	s
1	1	1	0	0	0
1	0	1	1	0	0
1	0	0	1	1	0
0	1	0	1	1	0
0	0	1	1	1	0
0	1	1	0	1	0
0	1	1	1	0	0
1	1	0	0	1	0
1	1	0	1	0	0
1	0	1	0	1	0

Inference :

When a cell gives a new clue, we use it to find binary combinations of its neighbors. This information is then combined with previously known combinations in our knowledge base in such a way that they do not contradict each other for common neighbors. This gives us a new set of combinations which is more accurate than before.

For example, if in the above example we explore d and it reveals 1

	a	b
x	c	3(s)
y	1(d)	e

Then the combination of its neighbors are :

c	e	x	y	where d = 0
1	0	0	0	
0	1	0	0	
0	0	1	0	
0	0	0	1	

We compare this with previously known combinations and discard all except where cde are 100, 001, 000. After concatenating all valid combinations we get :

a	b	c	d	e	s	x	y
1	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0
1	0	0	0	1	0	0	0
0	1	0	0	1	0	0	0
1	1	0	0	0	0	1	0
1	1	0	0	0	0	0	1

The combined result takes into account all scenarios that are possible. This is done by finding all valid situations which are consistent with every piece of information. So the program does deduce everything it can at any given stage.

Decisions :

Given a current state of board and knowledge, the program needs to decide which cell to search next with minimum risk of stepping on a mine and also with the aim to maximize the revelation of useful information. It does so by finding the probabilities of cells being a mine based on the set of combinations that have been deduced and then searching a cell the neighborhood of which is already present in the knowledge base.

In the state described in previous example, we can calculate the probabilities of cells being a mine using the combinations

P(a) = (1 + 0 + 1 + 0 + 1 + 1) / 6 = 4/6 = 0.666
P(b) = (0 + 1 + 0 + 1 + 1 + 1) / 6 = 4/6 = 0.666
P(c) = (1 + 1 + 0 + 0 + 0 + 0) / 6 = 2/6 = 0.333
P(e) = (0 + 0 + 1 + 1 + 0 + 0) / 6 = 2/6 = 0.333
P(x) = (0 + 0 + 0 + 0 + 1 + 0) / 6 = 1/6 = 0.166
P(y) = (0 + 0 + 0 + 0 + 0 + 1) / 6 = 1/6 = 0.166

Once we find the probabilities we then need to decide which cell to search next. This is done as follows :

1) Find the cell with minimum probability of being a mine (say p) . If this value is zero, we are certain it is a clear cell. Otherwise we still can't say if it is clear (which means we will have to take some risk)

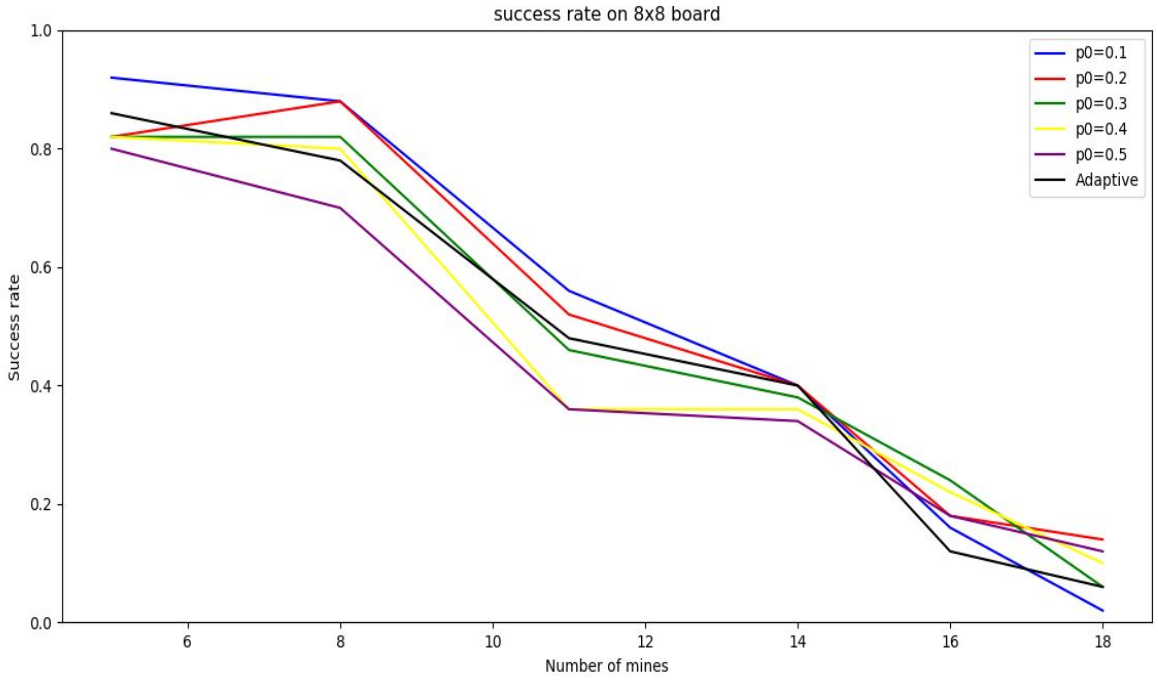
2) Next we compare p with a threshold value p_0 . If $p <= p_0$, we select the cell corresponding to p. If there are multiple cells having the same value p we go for the cell having the most number of identified cells (ie mines and clear cells) in its vicinity. This is done to reveal information which will be more useful in accurately determining remaining cells.

3) If $p > p_0$, none of the cells in our knowledge base are " safe enough " to search next. We now have to make a decision to select a cell we don't know anything about. Even now we expect the cell to reveal information which will help us make our knowledge base more accurate. So the cell which is closest to the identified cells (yet outside the knowledge base) is chosen and not just any random one.

   How do we fix the value of p_0 ?
   The value p_0 determines when our program selects a cell not already in the knowledge base to be searched next. This should happen when the value of p is high enough to be

too risky. It is important to note that if we search a cell outside the present knowledge base, there is a good chance that the information revealed would be less useful than searching some cell we have knowledge about (and this could be costly in future searches). So the value of p_0 should take this fact into consideration as well. If we already knew the number of mines present, it would be much more easy to fix the value of p_0.

We opted for two approaches to fix p_0 and have plotted the results below :
1) $P_0 = K$ (where k is a constant between 0 - 1)
This value should be high enough to be considered fairly risky and also account for possible loss of useful information.
2) Adaptive Method :
We tried determining the value of p_0 based on our knowledge base. The already searched portion of the mine can give us a general idea of how dense the mine is. Here again we used the deduced combinations to find Expected mines and divided it by the number of cells in the knowledge base.

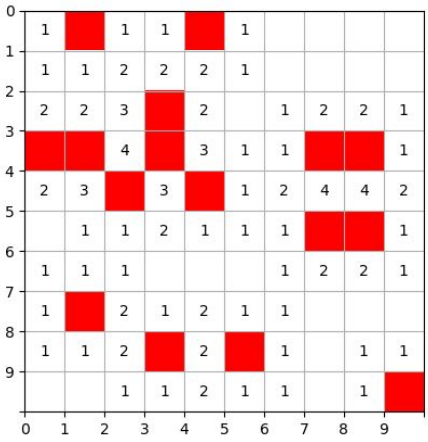


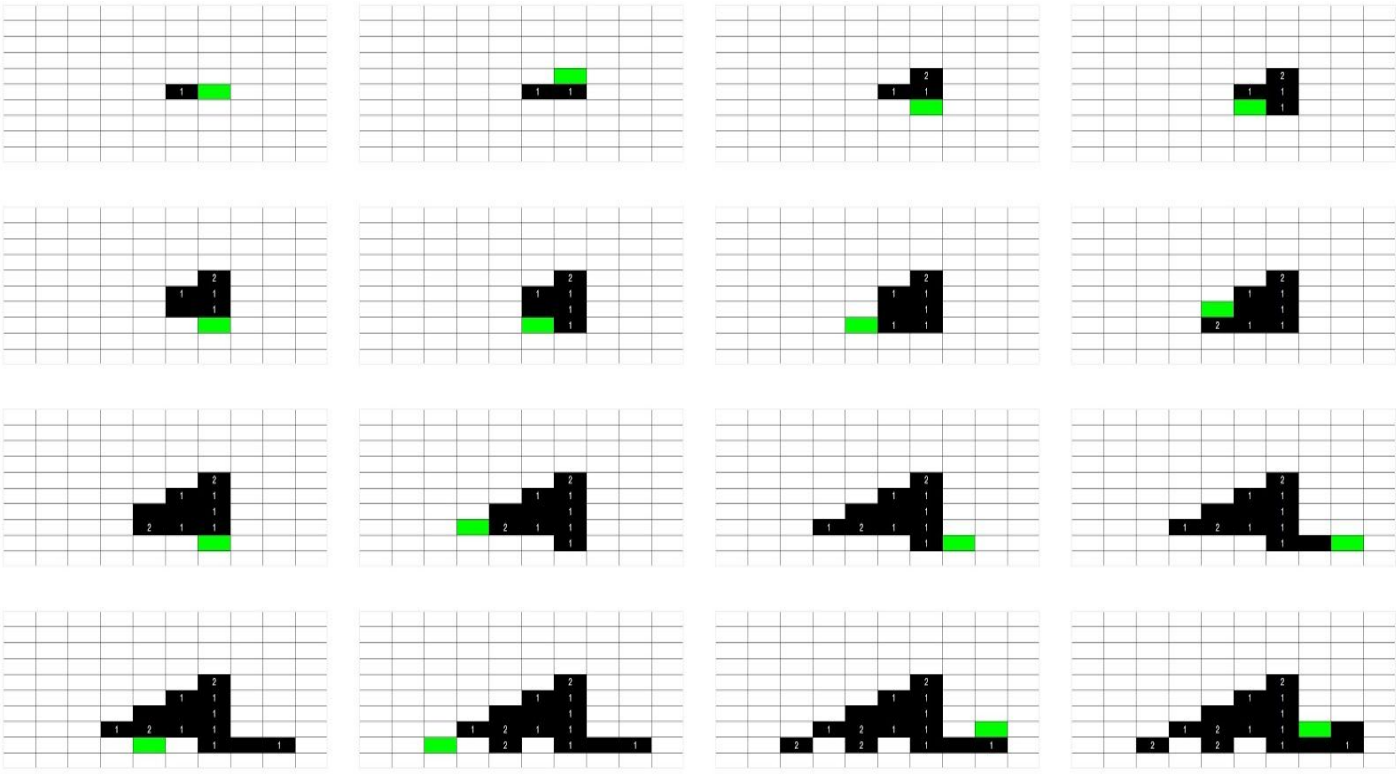
The results for different p_0 values show more or less the same trend. Which one performs better depends on the actual mine density.

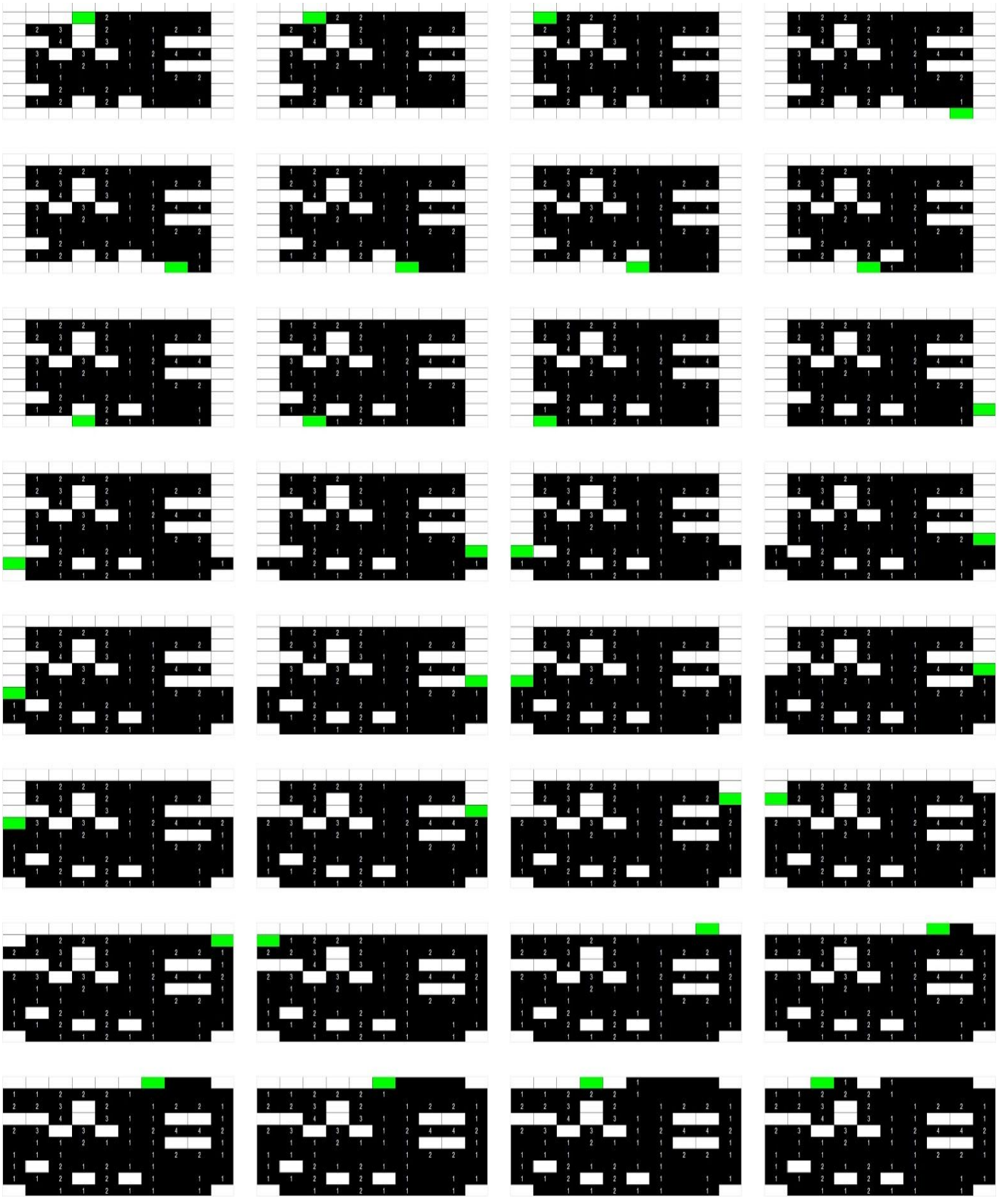
Performance :

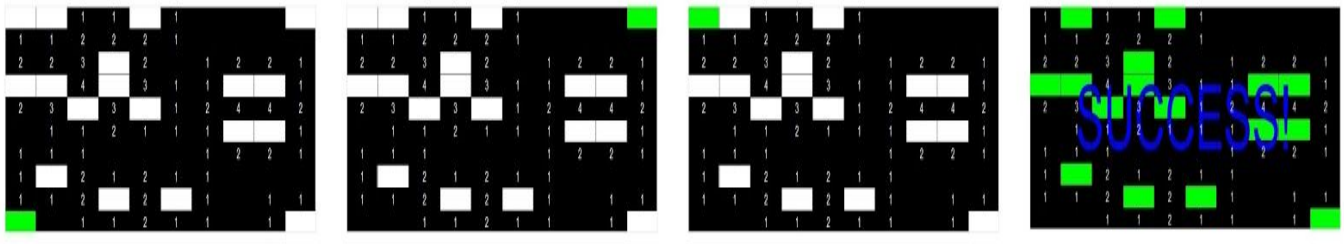
Shown below is a play by play account to success in solving a 10*10 mine

Mine to be solved

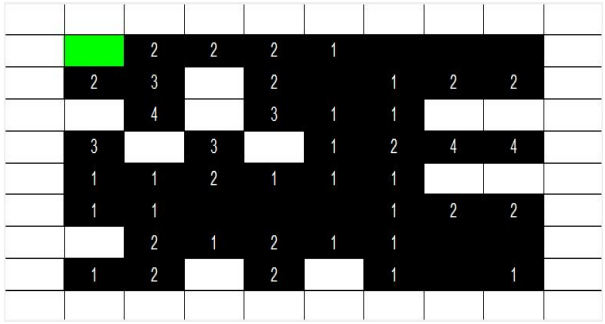


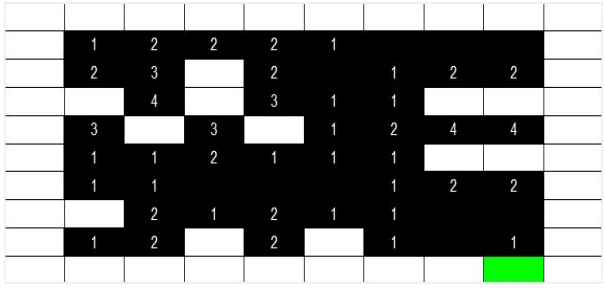






The 52nd move was a surprise.

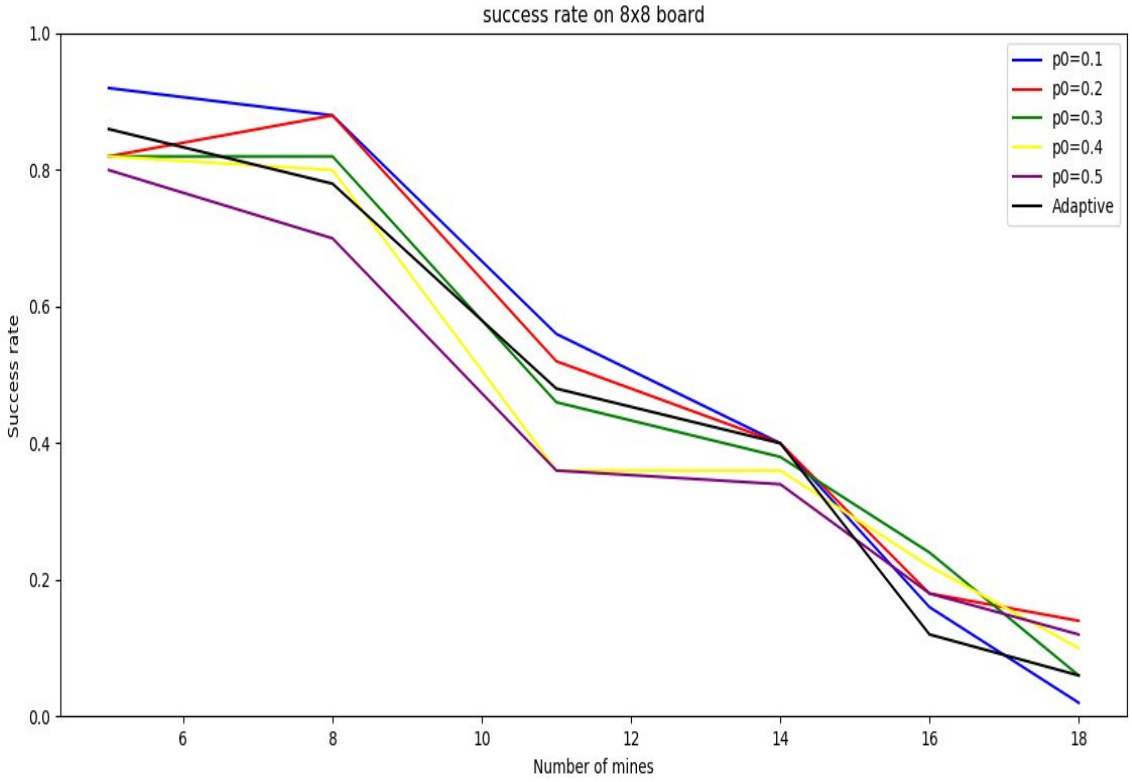




Here the program moves from the upper portion of the field to search a cell in the lowest row. It could do this because the program knew that this was the cell with the lowest chance of being a mine based on all the information it had collected thus far. A human would have probably found it difficult to find the lowest risk cell.

Performance :

Consider the graph for mines vs success rate shown below



It is evident that as the number of mines increases, chances of success goes on decreasing. For 8*8 field, it is approximately 40% for 14 mines, 20% for 16 mines and 12% for 18 mines. The program struggles in fields with more mines because at some point it needs to make a random move. In case of more mines, the chance of this random cell being a mine is higher and thus more failures.

Efficiency :

For larger sized mines with mine densities neither too high nor too low (say 0.4 to 0.6), the number of possible combinations (situations) that arise are quite large. This is because there are too many of both, unidentified mines and clear cells. The program is unable to determine the

identity of a large portion of the knowledge base with 100% certainty and the knowledge base just keeps growing without making many inferences.

This can be considered a problem specific constraint. The problem does not provide us enough information to conclude the identity of most of the cells. Our implementation mirrors the problem by including all scenarios in its knowledge base. We can make certain changes to make it more efficient in space and time but that will be at the cost of accuracy. For instance, we might simply assume a cell with p >= 0.75 to be a mine and p <=0.25 to be clear. This will reduce the number of possible situations drastically but obviously comes at the cost of making more wrong moves.

Improvements :

When we know the exact number of mines this information can be used to make more accurate decisions as it would be possible to establish the risk factor much better. When we did not know the number of mines, we practically had no idea about the cells outside our knowledge base. It was very difficult to determine when we should abandon our search within the knowledge base (concluding it to be unsafe) and look elsewhere. In other words, we can now find our threshold p_0 with a great deal of accuracy.

Say the number of mines is x in a m*n dimension minefield. We can use the deduced combinations to find the expected number of mines (y) within the knowledge base.

For example, if the combinations are

a	b	c	d
1	0	1	1
1	1	0	0
1	1	0	1
1	0	0	0

The expected number of mines is 2.25

x - y is then the expected number of mines distributed elsewhere. Let z = number of cells in knowledge base, then the probability of any cell outside the knowledge base being a mine would be : (x - y) / (m*n - z)

We add a constant ε to account for possible loss of useful information as discussed previously
Then our threshold $p_0 = (x - y) / (m*n - z) + \varepsilon$

As we progress in our search, and more mines get identified, the value p_0 becomes more accurate. This is because the expected mines ‘y’ would be much closer to the true value..

This will help the program perform much better even in high density minefields.

4 Bonus: Chains of Influence

• Based on your model and implementation, how can you characterize and build this chain of influence? Hint: What are some ‘intermediate’ facts along the chain of influence?

The chain of influences can be characterized and built in the following way:

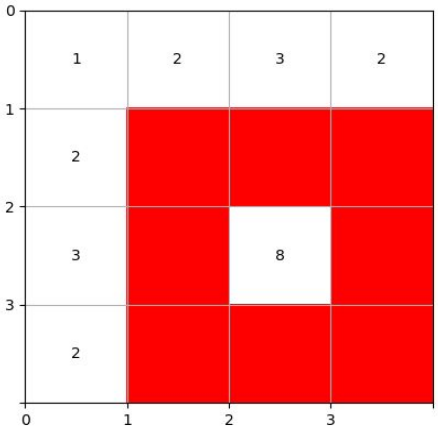
For any cell, ‘n’, that has been determined as a mine, it will be directly influenced by: i) a cell adjacent to n, ‘a’, that has already been revealed, and ii) the neighbors of a that have already been determined as clear.

For any cell, ‘n’, that has been determined as clear, it will be directly influenced by: i) a cell adjacent to n, ‘a’, that has already been revealed, and ii) the neighbors of a that have already been determined as mines.

And, the chain of influence of a cell inherits that of its parent. Therefore, the chain of influence can be tracked all the way to a source node in an acyclic graph. This node will either be the first node that was revealed or a node that was revealed at random, i.e. when all the cells in our knowledge base had a probability of being a mine greater than p_0.

The chains of influence for a simple 4x4 board (Fig 4.1) are shown in the DAG in Fig. 4.2

.Fig 4.1



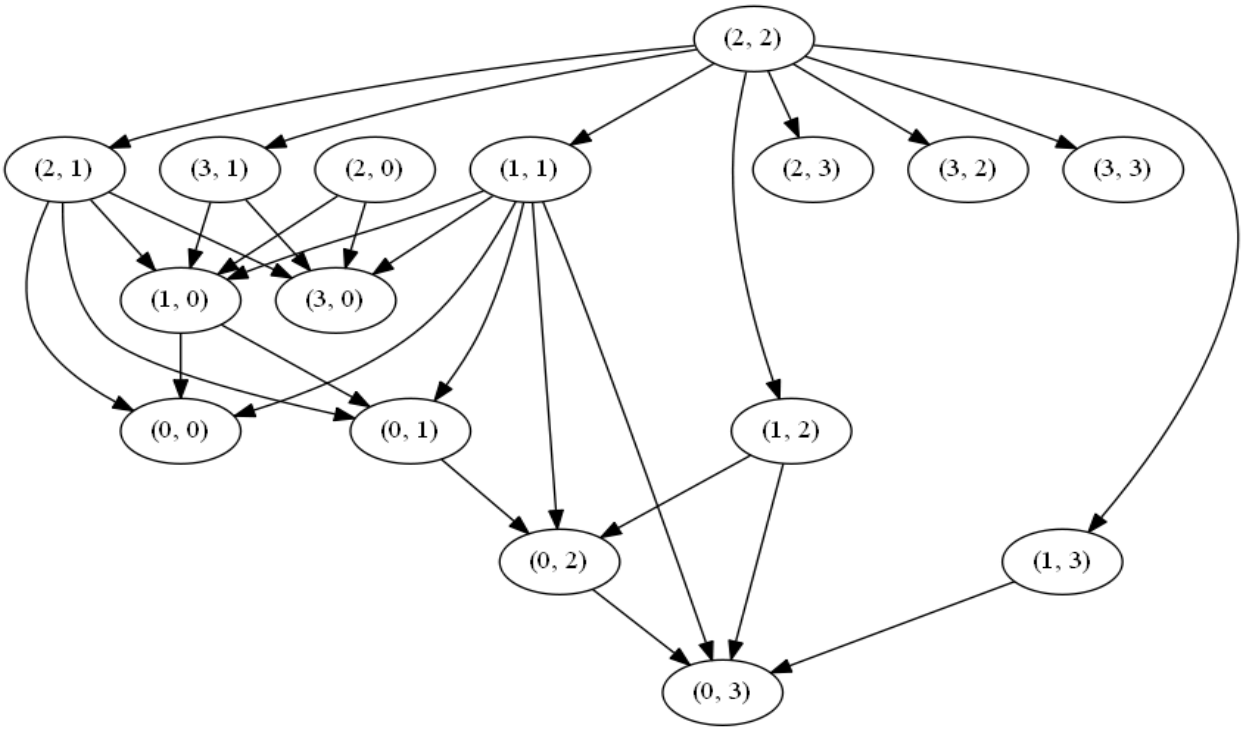


Fig 4.2. Chains of influence for board in Fig 4.1

• What influences or controls the length of the longest chain of influence when solving a certain board?

The answer to this is simply “the distribution of the mines”. The longest chain of influence, in particular, occurs when all the chains of influence for all the cells are part of a single connected component within a DAG having a single source node. In other words, this occurs when we don’t reveal any cell outside of our current knowledge base (i.e. guess a cell). This ensures that the connectivity of the component increases by not adding any new source nodes, thereby, leading to longer chains within it.

• How does the length of the chain of influence influence the efficiency of your solver?

As mentioned in the previous question, when no cell is revealed outside of our knowledge base we can expect longer chains of influence. The vice versa is expected to hold true as well, when we have long chains of influence lesser number of cells are guessed randomly. Therefore, at any point during the procedure there is at least one cell in our knowledge base whose probability of being a mine is lower than the threshold p_0, which in turn decreases the chances of running into a mine. Thus, the efficiency of our solves increases with the length of the chain of influence.

• Experiment. Can you find a board that yields particularly long chains of influence? How does this vary with the total number of mines?

100 trials were run for different number of mines ranging in 6,9,12,15,18 with the threshold p_0 = 0.5. The experiment revealed the following board (Fig 4.3) with long chains of influence. The longest chain achieved was: (4, 4) -> (5, 5) -> (5, 4) -> (4, 5) -> (4, 6) -> (5, 7) -> (6, 6) -> (7, 5) -> (5, 3) -> (6, 2) -> (6, 1) -> (6, 0) -> (5, 1) -> (4, 0) -> (5, 2) -> (3, 2) -> (2, 1) -> (1, 2) -> (0, 0) -> (0, 1) with a length of 20.

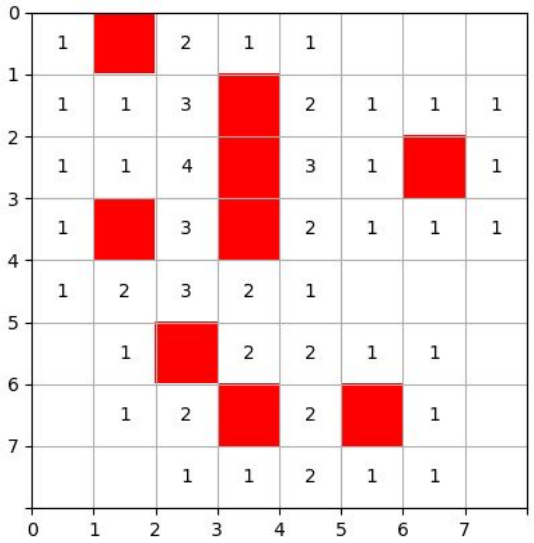


Fig 4.3. 8x8 board with long chains of influence

The following graph Fig. 4.4 shows how the length of the chains of influence varies with the number of mines. It is clear from the graph that the length of the chains increases with the number of mines. (This experiment included only those cases with number of mines less than a threshold M =18. This is to ensure that the maze remains solvable in most cases.)

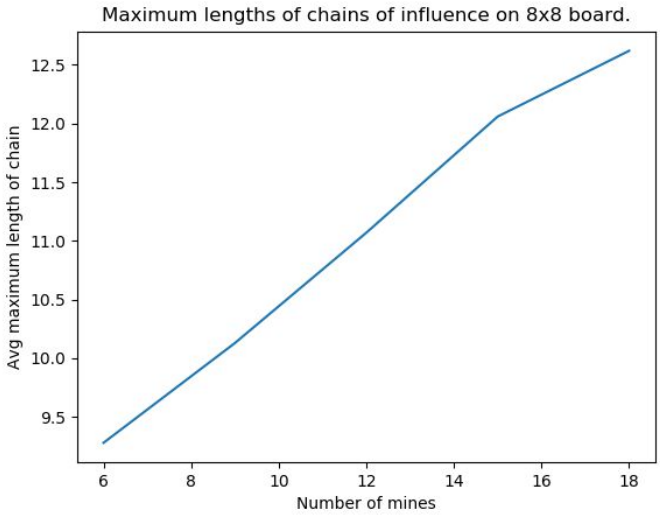


Fig 4.4 Maximum length of chain of influence v/s number of mines for 8x8 board

• Experiment. Spatially, how far can the influence of a given cell travel?

An experiment was run using the same set of data used in the first part of the previous question. Spatially, the influence of a given cell could travel as far from it as the point on the board that is farthest from it. In the case of the minefield presented below, the influence of the cell on the bottom right travelled all the way up to the cell on the top left of the field (i.e. along the diagonal).

One such long chain was achieved for the following 8x8 board (Fig. 4.5). Where the influence of cell (7,7) travelled up to (0,0), i.e. with a spatial manhattan distance of 14.

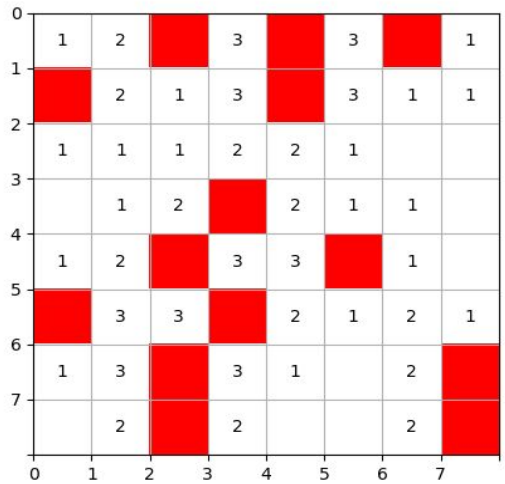


Fig. 4.5. 8x8 board with a cell having far reaching influence.

• Can you use this notion of minimizing the length of chains of influence to inform the decisions you make, to try to solve the board more efficiently?

No, we cannot use the notion of minimizing the length of chains of influence to solve the board more efficiently. Attempting to do so would require revealing cells whose probabilities of being mines are not the lowest in our knowledge base just to decrease the length of influence. This is an obvious case of making a bad decision which will increase the probability of running into a mine.

• Is solving minesweeper hard?

Yes, solving minesweeper is hard. In more cases than not, there is no deterministic solution to the board. At some point or the other we have to reveal a cell whose probability of being a mine is not 0. Going by this logic, the probability of our solver being successful can only be as good as 1 – p. Where p is the probability of the revealed cell being a mine on our riskiest move. (i.e. when the cell has the highest probability of being a mine of all the cells that were revealed).

Bonus - Dealing With Uncertainty :

Situation 1 : A clear cell reveals information about its neighbors only with some probability

In the usual case when correct information is revealed, we proceed as discussed earlier. However, in the case when a searched cell is blank we can include all possible combinations for the cells which we encounter for the first time (cells that are not present in the knowledge base) This is equivalent to not including the new cells in our updated knowledge base because adding all possible combinations does not provide any new useful information. So we might as well opt to not add them at all ie. do absolutely nothing in the case of a blank cell and simply proceed with searching another cell

We implemented the modified minesweeper for the uncertain situation. A comparison between efficiency of the normal case and uncertain case in solving a 8 * 8 mine with different mine densities is tabulated below. The probability that a mine will reveal information is kept 0.8 and 500 samples were analyzed for each case.

Mine Density = 0.1	Normal Minesweeper	Uncertain Minesweeper
* Success %	82.3	58.4
% of correctly identified cells in Failed Cases	58.2	54.7

* Neglected cases when failure was within first 5 moves

Mine Density = 0.15	Normal Minesweeper	Uncertain Minesweeper
Success %	80.1	26.2
% of correctly identified cells in Failed Cases	55.1	49

Mine Density = 0.20	Normal Minesweeper	Uncertain Minesweeper
Success %	45.9	15.2
% of correctly identified cells in Failed Cases	50.3	41.3

Mine Density = 0.25	Normal Minesweeper	Uncertain Minesweeper
Success %	39.3	6.5
% of correctly identified cells in Failed Cases	44.3	33.5

Mine Density = 0.3	Normal Minesweeper	Uncertain Minesweeper
Success %	29.7	3.7
% of correctly identified cells in Failed Cases	39.2	29.4

As we can see, the success percentage decreases drastically for higher mine densities in case of uncertain minesweeper whereas the normal minesweeper still performs reasonably well. For 0.1 mine density the success ratio is about 3:2, which drops to 8:1 for 0.3 mine density

Situation 2 : The revealed value is less than or equal to true value.

In this situation, the actual value can be any possible value greater than or equal to the revealed value. However, the conditional probabilities (the probability of a particular value being true after the cell reveals information) will be different. We find these conditional probabilities of each value being true and then find the corresponding “weighted combinations”. These weighted combinations will be combined with already deduced combinations in the knowledge base (which are weighted too). The final combinations are again normalized to give them their final weights. When finding the probability of any cell being a mine, these weights are taken into account.

For Example :

Consider the state below :

	a	b
	c	s(3)
	d	e

At this stage, our knowledge base looks like follows :

Weight	a	b	c	d	e
w_1	1	1	1	0	0
w_2	1	0	1	1	0
w_3	1	0	0	1	1
w_4	0	1	0	1	1
w_5	0	0	1	1	1
w_6	0	1	1	0	1
w_7	0	1	1	1	0
w_8	1	1	0	0	1
w_9	1	1	0	1	0
w_{10}	1	0	1	0	1

Next we proceed to reveal d, which turns out to be 6

	a	b
x	c	s(3)
y	d(6)	e
z	l	m

The true value of d can be 6 or 7. (Can't be 8 as s is known to be clear)

We find their conditional probabilities : P(6/6) = (1/6) / (1/6 + 1/7) = 0.542 (w_{11})
P(7/6) = (1/7) / (1/6 + 1/7) = 0.458 (w_{12})

There will be $^7C_6$ = 7 combinations corresponding to value 6 of weight w_{11}/7 each and $^7C_7$ = 1 for value 7 of weight w_{12}

Weight	x	c	y	e	z	l	m
w_{11}/7	0	1	1	1	1	1	1
w_{11}/7	1	0	1	1	1	1	1
w_{11}/7	1	1	0	1	1	1	1
w_{11}/7	1	1	1	0	1	1	1
w_{11}/7	1	1	1	1	0	1	1
w_{11}/7	1	1	1	1	1	0	1
w_{11}/7	1	1	1	1	1	1	0
w_{12}	1	1	1	1	1	1	1

These are then combined with our knowledge base so that only those combinations where common neighbors are identical are selected (c e in this example). The weights are multiplied to get the weights for these newly formed combinations.

Weight	a	b	c	e	x	y	z	l	m
w_1* (w_{11}/7)	1	1	1	0	1	1	1	1	1
w_2* (w_{11}/7)	1	0	1	0	1	1	1	1	1
w_3* (w_{11}/7)	1	0	0	1	1	1	1	1	1
w_4* (w_{11}/7)	0	1	0	1	1	1	1	1	1
w_5* (w_{11}/7)	0	0	1	1	0	1	1	1	1
w_6* (w_{11}/7)	0	1	1	1	0	1	1	1	1

$w_7 * (w_{11}/7)$	0	1	1	0	1	1	1	1	1
$w_8 * (w_{11}/7)$	1	1	0	1	1	1	1	1	1
$w_{10} * (w_{11}/7)$	1	0	1	1	0	1	1	1	1
$w_5 * (w_{11}/7)$	0	0	1	1	1	0	1	1	1
$w_6 * (w_{11}/7)$	0	1	1	1	1	0	1	1	1
$w_{10} * (w_{11}/7)$	1	0	1	1	1	0	1	1	1
$w_5 * (w_{11}/7)$	0	0	1	1	1	1	0	1	1
$w_6 * (w_{11}/7)$	0	1	1	1	1	1	0	1	1
$w_{10} * (w_{11}/7)$	1	0	1	1	1	1	0	1	1
$w_5 * (w_{11}/7)$	0	0	1	1	1	1	1	0	1
$w_6 * (w_{11}/7)$	0	1	1	1	1	1	1	0	1
$w_{10} * (w_{11}/7)$	1	0	1	1	1	1	1	0	1
$w_5 * (w_{11}/7)$	0	0	1	1	1	1	1	1	0
$w_6 * (w_{11}/7)$	0	1	1	1	1	1	1	1	0
$w_{10} * (w_{11}/7)$	1	0	1	1	1	1	1	1	0
$w_5 * w_{12}$	0	0	1	1	1	1	1	1	1
$w_6 * w_{12}$	0	1	1	1	1	1	1	1	1
$w_{10} * w_{12}$	1	0	1	1	1	1	1	1	1

These weights are then normalised using $w = (w_i) / \Sigma (w_i)$ to get the final weights w_1, w_2 etc. This is done because the sum of all weights won't be 1

Finally, we find the probability of any cell being a mine using these weighted combinations

Situation 3 : The revealed value is greater than or equal to true value

We follow the same method as the previous one with the only change being the manner in which the conditional probability is determined. Here for any revealed value any of its lower values could be true. The conditional probabilities are determined accordingly