

**Artificial Intelligence**

**Assignment #2**

**Submitted by:**

**Project Mate 1: Parul Puri (Ruid:181005947)**

**Project Mate 2: Nitin Mali (RUID: 176007106)**

**Project Mate 3: Karan Ahuja (RUID:1730007078)**

**Department of Computer Science**

**Submitted to:**

**Dr. Charles Cowan**

**Aravind Sivaramakrishnan**

**Group Member’s Contributions:**

|  |  |  |
| --- | --- | --- |
| Parul Puri | Nitin Mali | Karan Ahuja |
| Code for Minesweeper Solver | Code for Minesweeper Solver | Code for Minesweeper Solver |
| Write Up | Code for visualizations | Write Up |
| Mathematical Implementation | Formation of flow of implementation in the code | Mathematical Implementation |
|  |  |  |

**Description of implementation:**

* The current implementation of the code consists of solving the minesweeper board using intelligent tracking, constraint satisfaction and logic and satisfiability.
* We succeeded in exploring the cells which are clear and those which reveal a clue in the first move and display a state of the board.
* We tried completing the current implementation with logical inference. We succeeded in developing a knowledge base with equations of all possibilities of presence of mine depending on the values revealed by the clue cells.
* Further, we used the knowledge base and implied proof by contradiction to contradict a specific assumption of a possible mine on a cell. This led to inferring the position of a mine and thus a safe cell which can be queried for a value by the user.
* The only concern left was to recursively find the inferences for all the unexplored node because of the complexity of the data structure chosen. We have chosen a dictionary because of which recursively calling dictionaries of dictionaries had become cumbersome.
* For the current implementation, after getting the inference by proof by contradiction, we query the user for safe cells. The knowledge base and the base dictionary is updated with newly entered value by the user.
* Beyond this, we use intelligent tracking and logic and satisfiability to check the unexplored nodes depending on the value of explored cells.
* The code is runnable as a normal python file. The user needs to enter the data for the size of the board and then for entering value in a specific cell.

**Step by Step Implementation:**

* **Enter the size of board. For this implementation, we could model it for 4\*4 board.**
* **Enter the initial coordinates to the start the board from.**
* **The clear and clue cells open**
* **Based on the values of the clue cells, the equations of the possibility of presence of mine on the unexplored nodes are formed. Equations are formed on the basis DNF since they are easier to compute.**
* **Based on proof by contradiction, we assume mines at different locations and get contradiction, and hence can infer for safe cells. Due to the complexity of the chosen data structure (dictionary), we are unable to generate a generic inference code. Thus, for the current model of minesweeper only few out of unexplored cells are inferred using proof by contradiction.**
* **After getting inputs from the user on the safe cells, its values are updated in the base dictionary.**
* **Now for the remaining cells of the board, we use intelligent tracking to find mine on the cell which has total neighbors- explored cells = value of that cell. This newly inferred mine is also updated in the dictionary.**
* **Now we use satisfiability to find the cells for which the mines have been inferred but are unexplored. These remaining safe cells are asked for values by the user.**
* **The last left cells which can’t be inferred by any manner need to picked randomly.**

**Questions and Representation:**

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

In the program we represented the board in form of matrix. We used the numpy library in python to create a 2-dimensional matrix to store the value. Each element in the matrix corresponded to a particular cell of the mine. The number/value in a particular cell corresponded to the number of adjacent mines to that particular cell. Say a cell revealed 4; it meant that 4 of the surrounding cells are mines. If the cell contained 0, all the neighboring cells were clear and could be safely revealed. To represent the information knowledge the clue cells revealed we used dictionary in python. Dictionaries are nothing but unordered key value pairs. In our implementation, the key was the location of a particular cell (M10 i.e. cell 1, 0) and value corresponds to the number of mines around it. If the value of a particular cell is 1 it corresponded that one of its surrounding cells has a mine. When a particular cell revealed a clue, we stored all the possibilities of neighboring cells being a mine in the dictionary. The equation containing conjunctions and disjunctions of all possibilities after a particular clue was revealed was stored in the dictionary. We employed a dictionary of dictionary to address this. Whenever a particular clue was revealed the dictionary was appended accordingly.

*Approach towards the minesweeper problem:*

To work towards the minesweeper problem we employed the following approach. We first built a 4\*4 mine and tried to manually solve it in a way the computer/machine should solve. We made all the equations and understood how to proceed ahead. Our next step was to implement this in a code and then try to leverage it to a 10\*10 mine or more. We relied to use Distributive Normal Form rather than Conjunctive Normal Form. Going through the notes and theory we concluded that DNF is easy to implement compared to CNF.

* Inference: 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? 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?
* For the latest submission, we use intelligent tracking, logical inference and satisifiability as well. The code uses intelligent tracking to find the mines on the neighboring cells i.e. depending on the data revealed in the clue cells and the remaining unexplored nodes, the decision is made on whether a mine is found or not. Based on the inference made of whether a mine is detected a corresponding safe node is queried for a value by the user. We have also been able to generate the equations using DNF to predict possibilities of mine in the surrounding unexplored nodes. This data helps us in intelligent tracking by finding the number of unexplored cells given the neighboring nodes of the cell and the cells adjacent to it that have been explored. For logical inference, When a new clue is collected, based on the known data of neighboring cells we generate an equation for all the unknown cells in the form of conjunctions and disjunctions and that to the dictionary which is our knowledge base. In simple terms when a new clue is revealed we built the corresponding equations and append that in our knowledge base. We deduce reasonable amount of information needed to go ahead and work on proof by contradiction. We assured ourselves that we deduce reasonable amount of information by manually solving it in on paper first and then implementing the code on it. Using the information, we deduce we were able to solve for mines and go ahead. There are several things we could improvise on. We just generate the equation to our knowledge base. We envision that we could improvise this deduction process in the following way: Based on the new clue revealed we can compared the particular cell to neighboring cells and try to reduce the length of the equation by intelligently predicting the possibility of which cells can be mine and which cannot be mines. This can be done by comparing the deduced data with the known data and then adding it to the knowledge base. This reduces the size of the equation and thus the computations we need to do. One other way we think we could add more intelligence to our system is introduce CNF and use DPLL algorithm for satisfiability. Due to time constraints and limitations of our ability to use data structures efficiently we were not able to implement these ideas.

We used satisfiability to check whether he number of mines a clue cell revealed is equal to the number of explored neighboring mines. This helps us to efficiently and safely open the remaining unexplored cells if number of mines a cell reveals is equal to the number of mines in neighboring cells which are known.

* When a new clue is collected, based on the known data of neighboring cells we generate an equation for all the unknown cells in the form of conjunctions and disjunctions and add that to the dictionary which is our knowledge base. In simple terms when a new clue is revealed we built the corresponding equations and append that in our knowledge base. We deduce reasonable amount of information needed to go ahead and work on proof by contradiction. We assured ourselves that we deduce reasonable amount of information by manually solving it in on paper first and then implementing the code on it. Using the information, we deduce we could solve for mines and go ahead. In the current implementation, if we start by querying for a cell which is clear, hence the code opens all the surrounding cells which reveal clues. Then the equations for all the clue cells are formed which use DNF to assume the position of mine at one of the cell corresponding to the value revealed in the clue/parent cell. The equations are then used in the inference by assuming a mine at one of the unexplored cells. This cell is selected by iterating through the set of equations to find the cell with the most constraining value. This approach uses the fundamental concept of most constrained variable. It selects the cell which occurs in most equations and assumes it to be a mine. Based on this assumption we follow proof by contradiction to find if the assumption was incorrect. The implementation of proof by contradiction involves forming a dictionary of assumption and inferred value, then this inference is checked for the remaining equations in the parent dictionary and solves them to find if a contradiction exists. Based on the contradiction, the assumed cell is made sure of being safe and can be asked by the user for the value. We have found a contradiction for particular cells but the process of recursively finding contradictions and inferences for all the unexplored nodes would be a part of our future implementation. There are several things we could improvise on. We just generate the equation to our knowledge base. We envision that we could improvise this deduction process in the following way: Based on the new clue revealed we can compared the particular cell to neighboring cells and try to reduce the length of the equation by intelligently predicting the possibility of which cells can be mine and which cannot be mines. This can be done by comparing the deduced data with the known data and then adding it to the knowledge base. This reduces the number of equations and thus the computations we need to do. One other way we think we could add more intelligence to our system is introduce CNF and use DPLL algorithm for satisfiability. Due to time constraints and limitations of our ability to use data structures efficiently we were not able to implement these ideas.
* 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?

We keep a track of the most constrained nodes in our knowledge base i.e. the cells that appear most of the times in the knowledge base. We then use proof by contradiction for the most constrained node. As the most appearing cell has more chance to come in contradiction and thus we can logically infer. Certainly there are risks associated with it. We work on the most constrained nodes and thus expect no contradictions. It might happen at some cases that we face no contradiction and cannot land on an inference. Also this increases the computational time. This risk does not lead to any failure but just results in delayed computations. So, for now we just work on getting the right result but not on complexities. Depending on the contradiction and hence an inference made that a cell is a mine we move ahead by using this inference from the equation it was derived from and deciding if a safe cell is found. If a safe cell is derived, we query the user to enter a value for that cell. We do reach a point in the implementation that the equations will not lead to any conclusion and hence we will need to resort to probability to detect a safe cell which might lead to failure.

* 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?

For the current implementation which uses intelligent tracking to find a mine, there are no points in the program in which we find decisions that we don’t agree with. In our future implementation though when we reach a stage where the equations or assumption would not lead to any conclusion and the program will need to query the user for a cell based on probability, we might find a surprising decision.

* Performance: For a fixed, reasonable size of board, what is the largest number of mines that your program can still usually solve? Where does your program struggle?

In the current implementation our program will be able to solve a board of size 10\*10. The program can solve about mines corresponding to about 10% of the total cells on the board. The program will struggle for cases where number of mines increase because inference is not implemented completely.

* 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?

Our implementation is mostly DNF based, so as we keep exploring more and more cells, the size of the knowledge base will increase. This means the memory requirements will keep on increasing as we increase the number of cells or the size of the matrix. Again the time complexity increases as we have to iteratively search over the whole knowledge base. For each assumption we do for proof of contradiction we iterate through the whole knowledge base, hence as we get more and more data the time to logically infer increases. Mostly these are implementation constraints. We can address these issues by using CNF and DPLL. However as we work on limited size of minesweeper we thought a brute force version would suffice our problem.

* Improvements: Consider augmenting your program's knowledge in the following way - when the user inputs the size of the board, they also input the total number of mines on the board. How can this information be modeled and included in your program, and used to inform action? 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?

This information can certainly help to increase the speed to solve the minesweeper maze. We can keep track of number of mines explored and the number of mines remaining. So as we are exploring cells we can work on cells which have the most constraints. Having a prior knowledge of how many cells to solve we can track the progress of our work and easily terminate when we explore all the nodes. Also prior knowledge of number of mines helps us to play with probability. If we know that the number of mines in the maze is more we have to play more safely and accurately infer on the data. If we know that number of mines is less we can play with some calculated risks. This helps the minesweeper solver to solve more quickly. Logical inference with probability is always helpful to go ahead. In the current implementation, we have used this information of the total mines on the board to intelligently track the mines left and stop the program when there are no mines left on the board.

**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?

In our implementation we can build this chain of influence in the following manner: Our implementation is based on proof by contradiction. We use proof by contradiction to logically infer the knowledge about cells. After every logical inference of 2-3 cells, we can iteratively check trough all cells. In this checking we will explore the known neighboring cells of each particular cell to see if the number revealed by the cell matches the number of mines. If it matches we can set all the remaining unknown nodes as clear. Example: suppose cell (2,2) reveals the number 2. It means 2 of the neighboring 8 cells are mines. From 2,2 we will check through all the known cells and count the number of mines we know. If it matches 2 we can safely mark all other cells as clear. In our implementation the cells which are clear , open the clue cells surrounding it. This leads to formation of a huge chain of clue cells around the clear cells. Based on the information revealed in the clue cells then the further equations are formed which lead us to prediction of mines or safe cells.

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

Following factors influence the length of the longest chain of influence:

1. Current state of the board: as the number of known states in the board increases the data we have increases. This helps us to compare the neighboring cells and conclude with knowledge. We can imagine that if a board is 50% solved then we know that we have sufficient knowledge to look around neighbors.
2. Number of mines on the board: If the number of mines on the board are less many cells would be clear and possibly could also have 0 meaning all cells are clear. This can help build a long chain of influence.

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

The more the length the faster a solver could solve the game. Also it reduces the computations and the memory used during computations.

* 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?

Yes we can use this notion of minimal chain of influence to indicate our solver that the numbers of mines around are more and to play more safely.

* Is solving minesweeper hard?

Solving minesweeper becomes hard when the size of the board increases and the number of mines increases. In this case the algorithm needs to store a lot of data in the knowledge base and conclude safely to go ahead.

**Citations:**

* The code has been completely self-generated by first developing a mathematical implementation for a particular problem and then modulating it in the form of a code.
* For understanding the theory and the approach to solve the problem we saw some videos on logical satisfiability and constraint specification problem which are as follows:

<https://www.youtube.com/watch?v=lCrHYT_EhDs>

<https://www.youtube.com/watch?v=il20Q5tXp-A>

* The text book “Artificial Intelligence A Modern Approach Second Edition” was referred to understand the way constraint propagation and constraint specification need to be addressed to solve the minesweeper.