Minesweeper Solvers: Final Project Report

1 Abstract

Minesweeper is a popular stochastic single player puzzle game where the player attempts to uncover all cells on the grid which do not contain mines. Extensive research has been done to create agents capable of solving Minesweeper efficiently and accurately using multiple strategies. These strategies include machine learning, search algorithms, and logic based agents which exploit the known rules of Minesweeper. This paper explores the use of five heuristic based agents to solve Minesweeper boards of different dimensions and difficulties. The heuristics encompass random moves, neighborhood sum, minimizing mine probability, maximizing entropy of neighboring mines, and a combination of minimizing probability and maximizing entropy of neighboring mines. For each heuristic, 100 simulations of Minesweeper were run and the metrics win rate, computation time, and average number of moves were recorded for each difficulty.

2 Introduction

2.1 Problem Description

Minesweeper is a popular single player puzzle-solving computer game. The following descriptions are based on the playable Minesweeper games on the website *minesweeperonline.com*. At the start of each game, there is a blank grid of unrevealed cells. Beginner games have a 9 by 9 grid of cells, intermediate games have a 16 by 16 grid of cells, and expert games have a 30 by 16 grid of cells. Each cell can either hide a mine, store a value, or be blank. The number of mines is shown in the upper left corner of the game interface. Beginner games hide 10 mines, intermediate games hide 40 mines, and expert games hide 99 mines. Each mine is randomly planted in its own unique cell at the start of the game. The values that many cells store represent the total number of surrounding cells that contain mines (up to eight for the eight surrounding cells). If a cell is blank, that means that there are no neighboring cells that contain mines. Information about what a cell contains is unknown to the player until that cell is revealed. To reveal a cell (make a move), the player can left-click it on the grid. Other cells may be revealed along with the chosen cell depending on the hidden arrangement of mines.

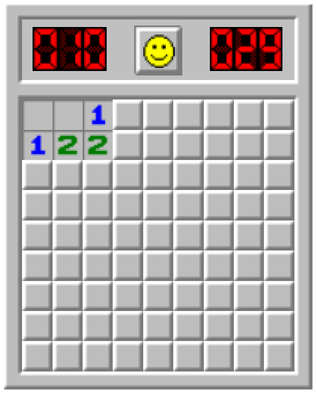
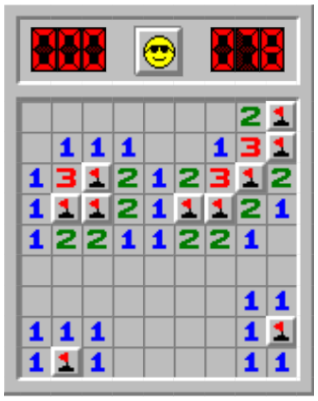


Figure 1: First move made in a beginner Minesweeper game.

Figure 1 above shows an example beginner Minesweeper board. The first move was made by the player in the upper left cell. Six cells were revealed. The blue *1* in one of the cells in the top row means that there is one mine hidden in the unrevealed neighborhood of that cell. Note that the number shown in the upper right corner of the game interface represents the total time elapsed during the game. The player can keep making moves until the game is over. To win, all of the cells that do not contain a mine need to be revealed. If the player reveals a mine, the game ends in a loss.


(a) Minesweeper win.

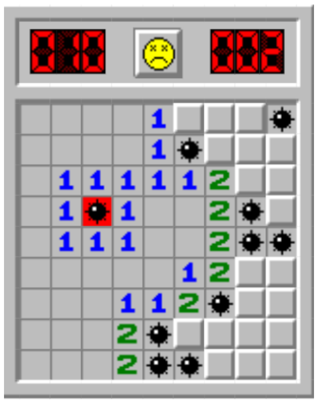

(b) Minesweeper loss.

Figure 2: Possible endings for a Minesweeper game.

Figure 2(a) above shows the final state of a winning Minesweeper game. The flags above each mine are optional. They can be placed by the player to help keep track of where the mines are. Figure 2(b) above shows the final state of a losing Minesweeper game. The mine highlighted with red was revealed by the player. The result shows the locations of the other mines [9].

2.2 Problem Interest

Artificial intelligence agents can solve Minesweeper in different ways, which makes the problem very interesting. Using logical analysis of the values of cells, many of the mines in a given Minesweeper game can be located for sure. However, there may be situations where it is impossible to know exactly where a mine is. Take the following game from *minesweeperonline.com*.

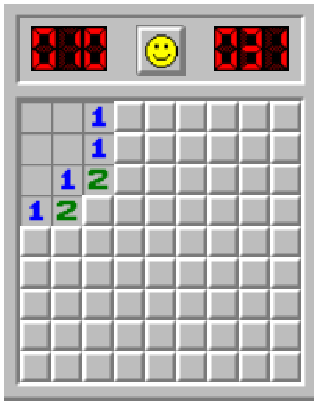


Figure 3: Unknown mine locations [9].

In Figure 3 above, the exact mine locations are unknown! Some sort of (educated) guess will need to be done.

The success of artificial intelligence agents when solving Minesweeper games can be measured in different ways, which also makes the problem very interesting. Success can be measured by the win rate, the solving time, or the number of moves. The size of the grid and the mine density can each be varied as well.

3 Related Work

3.1 General Search Algorithms

In the paper *Depth-First-Iterative-Deepening: An Optimal Admissible Tree Search* professor Richard E Korf. explores the workings of the IDDFS algorithm and its applications in search problems. The effectiveness of the IDDFS algorithm comes from it being able to combine the memory efficiency of a breadth-first search and the runtime of a depth-first search. By combining the two aspects of these algorithms, IDDFS becomes an efficient, optimal, and complete search algorithm on a finite search space. In the paper, Richard also explores the idea of combining IDDFS with a best-first heuristic search to decrease the number of nodes expanded [7]. Allowing the heuristic to guide the IDDFS and cut off at a certain depth limit based on the heuristic value prevents the search from looking at a path that is guaranteed to be suboptimal. This can also be connected to Minesweeper by using a heuristic to prevent the IDDFS algorithm from going down a path which is known to have a high probability of cells which will be mines. Instead it will cut off at a depth which has a high probability mine threshold, and continue searching in a safer location with a higher probability of safe cells. Then, as more cells are revealed it will become easier to determine which cells are safe or mines so the previously skipped cells can be safely revisited.

Various other search algorithms have been introduced to solve Minesweeper boards which are presented by Kasper Pedersen in *The complexity of Minesweeper and strategies for game playing.* The paper gives a general overview of Minesweeper as an NP-complete problem and then provides analysis and results for several Minesweeper solving strategies. One of the most effective strategies described in the paper was limited search which achieved win percentages of 91.7%, 64.3%, and 17.0% on beginner, intermediate, and expert boards respectively [10]. Limited search works by examining the "zone of interest" of an uncovered cell, which are all the cells in its proximity that can be used to determine the safety of the given cell. The strategy begins by assuming the cell contains a mine, and then constructing all possible configurations with no uncovered squares in the zone of interest. If no valid configuration can be found with the current assignment then backtracking occurs with depth-first search and new assignments are made. Once all configurations are exhausted a conclusion can be made on whether the given cell is safe or contains a mine. Using this strategy can also help decrease the amount of cells which need to be searched while still maintaining a high win rate which was an issue with the IDDFS approach. This search strategy is also effective because it uses the entire zone of interest to gain information instead of just adjacent cells which can help minimize guessing in certain instances. The increased use of information can also improve the search process because the heuristic function can rely on this information to guide the search.

A unique approach taken to solving Minesweeper with search algorithms can be found in *Combining Myopic Optimization and Tree Search: Application to MineSweeper.* In the paper, authors Michele Sebag and Olivier Teytaud treat Minesweeper as having multiple belief states which can be produced at each move based on the information that is gained after uncovering a specific cell. For each belief state, a belief state estimation can be used which samples the probability distribution on the possible Minesweeper board based on known information [12]. Using this approach, the authors combine multiple strategies such as a belief state solver and upper confidence trees which rely on Monte-Carlo simulations and performing tree searches [12]. With this approach and a slightly modified Minesweeper implementation, the solver is able to perform better than a CSP-based approach on all board dimensions tested in the paper [12]. This paper highlights how viewing the Minesweeper problem in a different light allowed them to move past the classical approach of using solely CSP's. In the future to try and improve the effectiveness of minesweeper solver strategies, it's necessary to move away from the standard approach. Instead, implementing new data structures and well known algorithms by manipulating the information that can be taken from the board can open the way for faster and more space efficient solutions. This research also illustrates how combining existing search strategies can help in developing a solver which is able to take advantage of each individual strategy's benefits and mitigate their weaknesses.

The A* search algorithm has also been utilized in a grid-like game similar to Minesweeper which is described in *Application of A-Star Algorithm on Pathfinding Game.* In the researcher's game, there is a character who must navigate the 2D grid world containing obstacles such as barriers with the ultimate goal of finding and protecting a tree object from any enemies [3]. This is done using the A* search algorithm with the Manhattan distance heuristic as the character traverses through the grid world based on the cost of a specific path. While traversing it also ensures that any illegal paths that will hit barriers or other objects aren't taken when finding the tree. After finding the tree the player also performs backtracking to the initial parent node in order to obtain the optimal path from the start to the goal node that doesn't contain any obstacles. From running the algorithm on various starting and goal positions, as well as different numbers of obstacles, they were able to conclude that the number of visited nodes and search time were directly proportional to the number of obstacles [3]. The findings of the research are important to consider when designing a Minesweeper solver as they highlight that A* search will be heavily limited on boards with a large number of mines. This means that it's crucial to utilize a heuristic that will mitigate visiting cells that are guaranteed to have mines or be in a position with many adjacent mines. Similarly, the idea of using backtracking alongside A* search could be valuable for a Minesweeper solver when it gains information when exploring the grid. This is because it may be able to determine if a certain cell is guaranteed to be safe or a mine based on cells it has uncovered after visiting the given cell. So backtracking would allow for the search to make more informed decisions based on newly revealed cells and improve performance if a heuristic is used which can exploit information gained over time.

3.2 Minesweeper Heuristic-Based Strategies

A heuristic strategy about considering the risk associated with each cell was tested by researcher Angela Huang in her project paper *Develop Heuristics to the Popular Minesweeper Game*. The main heuristic she looked into is minimizing the risk of choosing a cell with a mine by applying probability equations based on game state patterns. She wrote extensively to explain all the different combinations of known and unknown cells, and what their probabilities for revealing a mine are. She ran simulation trials to confirm her calculations, and found that the average simulation results were within +/- 0.0011 of the expected results. Note, Huang recommended to start the game by clicking random cells until about 50% of the board is revealed [6]. This is a (somewhat) simple example of a Minesweeper heuristic. Some look-up table could be used to hold game state patterns as keys and what cells to reveal (based on probabilities) as values.

Many different machine learning techniques were analyzed by researchers in *Fast Constraint Satisfaction Problem and Learning-based Algorithm for Solving Minesweeper*. Some of these ideas are outside the scope of this literature review, as they involve training machine learning models. However, they used one simple, but interesting heuristic: Manhattan distance. The researchers claimed that cells near the edge of the board hold more information about the variables used in their algorithm. So, if a cell is closer to the edge of the board (calculated by Manhattan distance), it should be more likely to be chosen. One of their techniques (DSScsp+DSS) was accurate 82.20% of the time. Coupled with the Manhattan distance heuristic, it was accurate 86.24% of the time. Similar results were found for another technique and itself coupled with the heuristic [13]. This is another (somewhat) simple example of a Minesweeper heuristic. However, this heuristic seems to depend on the solving algorithm used.

Many heuristic strategies for Minesweeper were studied by researchers in *Exploring Efficient Strategies for Minesweeper*. These include minimizing the probability of revealing a mine (P), maximizing the entropy of the number of mines around a cell (Q), maximizing the expected number of safe cells (E), and maximizing the probability of revealing at least one safe cell (S). These heuristics were combined to create even more heuristics: PS, PU, PE, PQ, PSE, PSU, PSQ, PSEQ, PSEQU, PSEUQ, and PSEU (the heuristic order matters). The researchers also discussed the "optimal" Minesweeper strategy (D*), but this takes a very long time to run. They ran some tests combining the heuristics with D* (D* was used over the heuristics as the endgame approached). It was found that combining heuristics increased the success rate greatly. PSEQ was the best, solving 39.616% of expert games. Combining this with D* increased the success rate to 40.06% for expert games. Note, the researchers recommended to always start the game by revealing a corner cell [14]. These are even more examples of Minesweeper heuristics. Combining heuristics and weighting them appropriately could produce an even more efficient solving strategy!

Another heuristic strategy was experimented with by researchers in *Optimistic Heuristics for MineSweeper*. This heuristic is that when two cells have the same probability of having a mine, it is better to choose the one that is as closest to the frontier between unrevealed and revealed cells. They claimed that choosing the cell with the least probability of hiding a mine is efficient, but not optimal, as there can be ties. They implemented a solver with Minesweeper as a CSP (CSP), a solver using the tie-breaking heuristic (HSCP), and a solver using both of these strategies (OH). They found that for expert games, CSP solved 34%, HCSP solved 38.1%, and OH solved 38.7%. Note, the researchers recommended to always start the game by revealing a corner cell [2]. This is an easily overlooked example of a Minesweeper heuristic. It is important to have a tie-breaking strategy so the solver always knows what action to take next.

3.3 General Minesweeper Strategies

The known rules and common strategies for Minesweeper were explored by Professor Pal Rujan in *The Minesweepers' Bayesian Guide to Survival.* He started by extensively explaining how the Minesweeper board is initialized, what the value in each cell represents, and how logic can be used to reveal safe cells. He decided to represent the game board as a graph, where each cell is a node and the distance between cells is the number of edges between their nodes. Knowing this setup, he introduced some strategies. One strategy is using a 3 x 3 look-up table when game state patterns are recognized. This approach relies on the average bomb density being at or below 0.2. As the bomb density increases, another strategy is to calculate the probability of each node having a mine, and revealing the node with the lowest. One interesting variation of Minesweeper that Rujan brought up is that there could be "noisy code," where the code for the game could miscalculate the value of a node next to a mine. In this case, a strategy is dynamic programming. He went into depth explaining how complex matrices and summations can be used to play the game. In the case where the player cannot be sure of how the game is coded, a final strategy is modeling the slot-coding process (coding how the node values are calculated) and the priori distribution of bombs [11]. This shows that twists can be added to Minesweeper. As the game becomes more unknown or abstract, new strategies can be implemented to solve the puzzle.

The D* optimal strategy for Minesweeper that was mentioned earlier was first tested by researcher Shahar Golan, who wrote about the counting problems for Minesweeper in *Minesweeper on Graphs.* He took the approach that Minesweeper could be represented as a tree or graph. He described the Minesweeper consistency problem as: is there a legal (all uncovered cells have the correct values) assignment for a given Minesweeper grid? He described the Minesweeper counting problem as: what is the number of legal assignments to for a given Minesweeper grid? Finally, he described the Minesweeper constrained counting problem the same, but the total number of hidden mines is known. Through logical and mathematical proofs, Golan proved that the constrained counting problem on graphs with bounded tree width can be solved in polynomial time [5]. This shows the importance of choosing a data structure to represent Minesweeper. Different approaches could be more appropriate depending on the data structure.

Key components of creating a Minesweeper solver and the single point strategy are described by David Becerra in *Algorithmic Approaches to Playing Minesweeper.* Before explaining any solving approaches, Becerra explores the underlying importance of the first move in a Minesweeper board as well as the existence of guessing. In terms of the first move, Becerra mathematically reasons through how revealing a corner square is the best first move because it has the highest probability of having zero neighboring mines [1]. This is because it has the least amount of directly adjacent neighbor cells which maximizes its probability of having zero mines next to it. Due to the fact its oftentimes impossible to make completely deterministic moves for a given Minesweeper board, guessing must be considered when designing a solver. The solution presented by Becerra is to calculate the probability a given cell contains a mine given the information on the grid, and pick the cell with the least known probability. Understanding the role of the first move and guessing is crucial when building a solver as they can heavily determine the effectiveness of certain algorithms. The solving strategy presented is single point which is when an algorithm considers the constraints only one cell has on all of the neighboring cells. By combining known constraints it would become possible to determine which cells are safe and which cells have mines. However, due to the strategy's simplicity and inability to use all known information it performs poorly on intermediate and expert Minesweeper boards [1]. These findings illustrate the importance of creating a solver which is able to use as much board knowledge as possible while minimizing the time it takes to determine the safety of a cell. Finding this balance prevents the solver from making misinformed decisions about what cell should be uncovered next and making unnecessary guesses. The paper also highlights that it's crucial to develop a solver which considers the probabilistic conclusions that can be made about the state of a cell. That is because using these probability calculations maximizes the chances the solver can avoid losing by guessing which is a common occurrence on larger dimensions boards.

One of the most recent approaches to creating a Minesweeper solver uses large language models as described in *Assessing Logical Puzzle Solving in Large Language Models: Insights from a Minesweeper Case Study.* Researchers Yinghao Li, Haorui Wang, and Chao Zhang wanted to test the reasoning ability of a LLM by analyzing whether models such as the GPT-4 model would be able to identify and avoid the location of mines on a simple Minesweeper board [8]. One of the main issues the researchers ran into however, was finding a representation of the Minesweeper board which the LLM could comprehend. Similar to previous research findings, determining how to represent the Minesweeper board such as through a tree or grid heavily influences the effectiveness of the underlying algorithm being used. One of the ways the researchers overcame this was by providing either a table or coordinate board representation with annotations and giving the model the basic rules of Minesweeper. After testing the model on 5x5 boards with 4 mines, despite not being able to fully solve any Minesweeper games, it was able to flag 30% of mines in the table board representation [8]. However, the most beneficial finding of the research was that the LLM was prone to making blatant logical errors when trying to decide whether a certain cell was safe or a mine. Even though the model was given the Minesweeper rules, it would still incorrectly reason into a position where 2 mines would be adjacent to a cell that has 1 as its uncovered value [8]. Despite these limitations, the ability to use LLM to try and reason through Minesweeper the same way a human would is a substantial advancement in the realm of Minesweeper solver and opens the door to future research. The considerations of a LLM Minesweeper solver are also applicable to improve other Minesweeper algorithms such as how to represent the Minesweeper board and allow for logical reasoning to take place as more information is gained.

3.4 Summary

For general search algorithms, it should be known that (somewhat) simple agents can use IDDFS, backtracking, Monte-Carlo, CSP algorithms, A*, or combinations of these to solve puzzle games like Minesweeper. For Minesweeper heuristics, it should be known that there are many heuristics that can be used to help solve Minesweeper: minimizing the probability of revealing a mine, Manhattan distance, combining heuristics, revealing cells near the frontier of unknown cells to break ties, and many more. For general Minesweeper strategies, it should be known that new solving strategies can be implemented as the Minesweeper rules or representation change, solvers should take into account as much board information as possible when making a move, and LLM's can solve Minesweeper (but sometimes make simple logical errors along the way).

4 Heuristic Approaches

The general approach was to implement a Minesweeper game and a game loop. Inside the game loop, different Minesweeper agents with different heuristics could be chosen to attempt to solve the game. The first agent reveals cells at random until the game is over. The second agent reveals the cell with the smallest neighborhood sum. The third agent reveals the cell with the minimum probability of having a mine. The fourth agent reveals the cell with the maximum entropy for the distribution of the number of neighboring mines. The fifth agent builds onto the third and fourth by combining their heuristic strategies. Each of these agents was tested and the results were compared.

4.1 Random Move (#1)

The first agent that was tested used a heuristic that returns the coordinates of a random cell to reveal. It is very simple. First, the agent gets the list of unrevealed cells for the current given state. Next, it picks a random index. Lastly, it returns the coordinates of the corresponding cell to the random index from the list of unrevealed cells.

4.2 Neighborhood Sum (#2)

The second agent that was tested used a heuristic that returns the coordinates of the unrevealed cell which has the lowest neighborhood sum. The neighborhood sum is defined as the sum of all the status values of revealed cells adjacent to the unrevealed cell being looked at. This heuristic is meant to emulate the strategy of a very novice Minesweeper player who will observe the board and decide to reveal cells based solely on their neighbors and not patterns. So, a cell that is adjacent to revealed cells with very small status values will be deemed safer than a cell that has adjacent cells with higher status values.

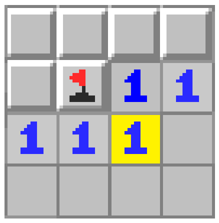


Figure 4: A case where the heuristic correctly avoids revealing a mine.

To achieve this, the neighborhood of each revealed cell is observed and the neighborhood sum of all unrevealed cells within that neighborhood is calculated, and the cell with the smallest sum is tracked. As in the case of Figure 4, the cell containing the mine will be avoided as it has a neighborhood sum of 4 and instead the cell above it with a neighborhood sum of 1 will be revealed. Only unrevealed cells that are in the neighborhood of a revealed cell are observed to eliminate iterating through cells that have a neighborhood sum of 0.

4.3 Minimizing Mine Probability (#3)

The third agent that was tested used a heuristic strategy that returns the coordinates of the cell with the least probability of holding a mine. The algorithm for this heuristic strategy was posed by researchers in *Exploring Efficient Strategies for Minesweeper*. The algorithm first creates S, which is a set of combinations that is helpful for solving for the mine probabilities.

```
$S \leftarrow \{s | s \in \mathcal{P}(C), \|s\| = m\}$
for all $b \in B$ do
    $S \leftarrow \{s | s \in S, \|s \cap U(b)\| = f(b)\}$
```

In the part of the algorithm above, *P(C)* represents the set of combinations of C, which represents the set of unrevealed cells. m is the total number of mines in the minefield. B represents the set of revealed cells. *U(b)* represents the set of the neighborhood cells of cell b. Finally, *f(b)* is the number of neighboring mines of cell b. The next part of the algorithm uses S to calculate the probability *p(b)* of each cell having a mine.

$$p(b) = P(b \in M) = \frac{\|\{s | s \in S, b \in s\}\|}{\|S\|}$$

In the part of the algorithm above, M represents the set of all cells containing mines [14]. As all the mine probabilities are calculated, they are compiled into a list. The coordinates of the cell corresponding with the minimum of these probabilities is returned. If there is more than one cell with the minimum probability, the coordinates of a random cell from these is returned.

4.4 Maximizing Entropy of the Distribution of the Number of Neighboring Mines (#4)

The fourth agent that was tested used a heuristic strategy that returns the coordinates of the cell with the highest entropy of the distribution of the number of neighboring mines. The algorithm for this heuristic strategy was also posed by researchers in *Exploring Efficient Strategies for Minesweeper*. The algorithm also uses S to help calculate the entropy of the distribution. After getting S, the distribution of the number of neighboring mines *p(b,n)* is calculated.

$$S(b,n) = \{s | s \in S, b \notin s, U(b) \cap s = n\}$$
$$p(b,n) = P(f(b) = n) = \frac{\|S(b,n)\|}{\|S\|}$$

In the part of the algorithm above, n is the value corresponding to the iteration number of the summation that is discussed next. The next part of the algorithm uses *p(b,n)* to calculate the entropy *q(b)* for each cell.

$$q(b) = -\sum_{i=0, p(b,i) \neq 0}^{\infty} p(b,i) \log_2 p(b,i)$$

In the part of the algorithm above, the summation is calculated from i equals zero to infinity. It skips iterations where *p(b,i)* equals zero [14]. To make the algorithm possible, i was chosen to max out at the length of *U(b)* plus the total number of mines. Similarly to minimizing mine probability, as all the entropy values are calculated, they are compiled into a list. The coordinates of the cell corresponding with the maximum of these entropy values is returned. If there is more than one cell with the maximum entropy, the coordinates of a random cell from these is returned.

4.5 Combining Heuristics (#5)

The fifth agent that was tested used a heuristic strategy that returns the coordinates of the cell with the least probability of holding a mine and the highest entropy of the distribution of the number of neighboring mines. The idea for combining heuristics like this also came from researchers that wrote *Exploring Efficient Strategies for Minesweeper*. Overall, this algorithm first finds the cells with the least probability of containing a mine. From those cells, it returns the coordinates of the cell with the highest entropy. The order here matters [14]. It calculates the probabilities the same way as the third agent, and it calculates the entropy values the same way as the fourth agent.

5 Experiments and Results

5.1 Experiment Design

Prior to testing each of the heuristics, base Minesweeper code in Java from previous coursework at the University of Minnesota was translated into Python. This translation was done to simplify the process of incorporating code needed for the heuristics and agents. The board was represented as a two-dimensional array of cell objects with each cell having two main attributes, status (mine or value) and revealed (true or false) [4]. Along with the board, functionality was available to accept user input, reveal cells, and evaluate the field to check if the game has finished as a win or loss. The game was playable through the terminal and provided an update of the board after each coordinate input and the total amount of moves made. To make the game compatible with the proposed experiments, when the minefield is initialized at the beginning of each game the first move is automatically made in the upper left corner (0,0). This meant that the initial state already had the first move made and allowed an equal start for all tested agents. The top left corner was chosen specifically as the standard for all agents because of the increased performance it showed in simulations mentioned in papers such as, *Exploring Efficient Strategies for Minesweeper.* For all difficulty boards the study found that revealing a corner cell as the first move can improve the win rate, and in some cases by over 3% in comparison to revealing a non-corner cell [14].

The overall implementation used for Minesweeper in this paper is a game loop where the agent chooses the coordinates of a cell to reveal that is returned by the given heuristic and then takes the action. This process is repeated until the game loop "terminates" by either the agent losing by revealing a mine, or winning by revealing all the safe cells. Each of the five heuristics described previously were coded as separate functions that performed the necessary calculations to determine the cell which should be returned to the agent. Inside of the individual heuristic functions, helper functions were called to attain the board's current revealed and unrevealed cells. Python's built-in combination function was also used to generate all of the combinations needed within set S for heuristics #3 and #4.

To test the heuristics, either 40, 60, or 100 games of Minesweeper were simulated with an agent using each heuristic on 3 different Minesweeper boards. This was implemented as a function which takes in the desired heuristic, number of games to be simulated, board dimensions, and number of mines on the board as parameters. The number of simulations was chosen to be 40, 60, or 100 to prevent memory errors while having a large enough sample size to draw conclusions from the results. This is because for larger board sizes and more complex heuristics, specifically heuristics #4 and #5 for the board of dimensions 6x6 weren't able to be ran for all 100 simulations without timing out in the middle of completing all the simulations. So instead, 40 simulations were run for heuristic #4 and 60 were run for heuristic #5. For each experiment, the designated heuristic, number of simulations, and board logistics were passed in and the body of the function would play out the games while tracking the number of wins and moves made. Initially, the logistics of the 3 boards were going to be the same as the beginner, intermediate, and expert Minesweeper boards. However, when running the simulations on these larger dimension boards, timeouts and memory errors occurred, so the decision was made to change the board dimensions. The first board became 4 by 4 with 2 mines, the second board became 5 by 5 with 4 mines, and the third board became 6 by 6 with 6 mines. These dimensions were picked because they avoided memory and timeout errors while maintaining a similar ratio of the number of mines to total cells. For Minesweeper's beginner, intermediate, and expert boards this ratio is approximately 0.123, 0.156, and 0.206 respectively. For the 3 board dimensions and mine combinations used in the experiments, the ratios are approximately 0.125, 0.15, and 0.167 respectively. By keeping these ratios relatively similar, the objective was to have the results of each simulation be accurate representations of standard Minesweeper boards despite the difference in dimension.

The tests for each heuristic were run on an A100 GPU using the University of Minnesota's MSI software. The separate simulations for each heuristic and board combination were queued as separate slurm jobs and the metrics of win rate and number of moves were returned when complete. In the case a GPU wasn't available, the tests were run on a cpu and the same metrics were recorded. A GPU was used when possible in order to speed up the simulation process and run the simulations in parallel if possible. The metrics of win percentage and average number of moves per simulation were chosen to analyze because they provided insight into each heuristic's accuracy and efficiency. Win percentage would be indicative of a heuristic's accuracy and whether it's a genuine solver for Minesweeper boards. The average number of moves per simulation would indicate if a heuristic is a viable Minesweeper solver because a heuristic that needs to perform numerous moves to attain a result on a smaller dimensional board won't be effective for larger boards. By considering both of these metrics it would be possible to identify the connection between win rate and number of moves, and whether there is a trade-off or correlation between them.

5.2 Results

Table 1 illustrates the results of running the 40, 60, or 100 simulations of Minesweeper using each heuristic on the 3 different board dimensions. The second pair, third pair, and fourth pair of columns correspond to the results of running the heuristics on the 4x4, 5x5, and 6x6 boards respectively. The win percentage was calculated by taking the total number of recorded wins and dividing by the number of games

Heuristic	Win Percentage	Avg. # of Moves	Win Percentage	Avg. # of Moves	Win Percentage	Avg. # of Moves
#1	15%	2.26	4%	3.18	0%	3.39
#2	47%	2.35	11%	3.80	4%	4.27
#3	92%	3.27	58%	5.10	40%	7.66
#4	26%	2.29	6%	3.47	2.5%	4.20
#5	86%	2.98	54%	4.69	32%	5.98

Table 1: Results of heuristic strategies.

played, and multiplying by 100 to convert to a percent. The average number of moves was calculated by taking the total number of moves performed over the designated number of simulations, and dividing by the amount of games played. Although overall time spent running the simulations wasn't recorded, there was a drastic increase in the simulation runtime as the board dimensions increased. In the case of the 4x4 board the runtime spanned only a few seconds regardless of the heuristic, whereas the total simulation took hours for multiple heuristics on the 6x6 boards. However, this extreme increase in runtime was only apparent in heuristics #3, #4, and #5 as heuristics #1 and #2 still took seconds to complete the 100 simulations. The results of heuristic #4 and #5 on the 6x6 board are taken with 40 and 60 simulations respectively due to the timeout errors which occurred when attempting to run 100 simulations initially.

6 Analysis

From the results shown in Figure 1, the best performing heuristic across all the boards in terms of accuracy was heuristic #3 which had win percentages of 92%, 58%, and 40% for the 3 board difficulties respectively. In comparison to the next best heuristic, heuristic #5, it performed around 5% better for each board difficulty. In comparison to all the other heuristics, it had more than double and triple the win percentage indicating its higher accuracy. However, along with the higher accuracy, heuristic #3 required more moves on average than all other heuristics. This ranged from approximately 1-5 more moves per simulation based on the board difficulty in comparison to the other heuristics. This positive correlation between win percentage and average number of moves is observable for all heuristics regardless of the board difficulty. Specifically in the case of the best performing heuristics, heuristic #3 and heuristic #5, it illustrates that an effective heuristic will be able to perform more moves as it solves the board. One of the main reasons this could be the case is because a more complex heuristics which relies on probability calculations is able to gather more information about the board. As opposed to heuristics #1, #2, and #4 which either rely solely on guessing or possibly faulty information about the neighboring cells, heuristics #3 and #5 utilize probabilistic calculations when figuring out safe cells. By incorporating probability into the heuristic, it's able to determine for certain when a cell has to be avoided, and in cases where it's impossible to know for certain, using probability maximizes the chances of still picking a safe cell. This is done by using the entire context of the board rather than just cells which may be adjacent to a certain uncovered cell which explains the lower performance of the remaining heuristics. However, there is a consequence in using more information which is possible runtime issues as seen by heuristics #4 and #5 as the more complex a heuristic is it needs to extract more information from the board. This can result in memory issues and lengthy simulations which can make the heuristics extremely difficult to use as Minesweeper solvers in a realistic application.

When comparing heuristics #2 and #3 to #4, the results illustrate that the complexity of a heuristic isn't a necessary indicator for it's success. In terms of computational cost and difficulty, heuristic #4 was far more complex than heuristic #2 and heuristic #3 but still performed worse than both for all board difficulties. When comparing heuristic #4 to purely guessing in heuristic #1, the results in the larger dimension boards is marginally better despite the extensive calculation and time that heuristic #4 required. Especially when considering that both heuristics #2 and #4 utilized calculations based on neighboring cells, it highlights the importance of a heuristic which manipulates information from the board properly. In the case of heuristic #2, it utilized the neighboring sum of cells which served as a better indicator for avoiding cells in comparison to the entropy calculation. This could be caused by the fact that in heuristic #2, any cells which aren't part of a neighborhood of already revealed cells are skipped and their neighborhood sum isn't calculated. Whereas for heuristic #4 the neighborhood of all unrevealed cells are considered which could result in misleading calculations about whether a certain cell is safe because of the lack of information that is used to reach a conclusion about a cell's safety. The absence of a relationship between complexity and accuracy is also shown by heuristic #5 which combined the 2 most complex heuristics, performing worse than heuristic #3 by itself. Despite being the heuristic which is the most computationally complex and requiring the most time to complete its simulation, it performed worse than heuristic #3 by 6%, 4%, and 8% for the 3 board dimensions respectively. This result demonstrates that providing the agent with a heuristic which considered more factors about what cell to pick, didn't necessarily mean it was able to make a better decision. In the case of heuristic #5 the results show that including the consideration of entropy masked the effectiveness of the probability calculation. The main difference between the two calculations is that the probability calculation considered the information gained from the entire board whereas the entropy calculation was based on the context of a cell's neighborhood. So, by combining these two pieces of information it diminished their effectiveness and misled the agent when picking a cell to reveal instead of giving it more insight. This principle could be applied to multiple heuristics that are implemented and shows that it's important to consider how the information heuristics provide to an agent can either conflict or support the agent in making a better decision.

The drastic decrease in win percentage across all heuristics between the different boards show how their effectiveness is heavily dependent on the board's dimensions. In previous research done on Minesweeper solvers such as in *Exploring Efficient Strategies for Minesweeper*, there is usually a slight decrease in a heuristic's performance from the beginner board to intermediate board which is around 3% and a more extreme dropoff occurs between the intermediate and expert boards [14]. In the case of this experiment's results, the accuracy of all heuristics decrease drastically between the 4x4 and 5x5 boards as they range from 11% to 36%. The decrease is less dramatic between the 5x5 board and 6x6 board, but there is still a large gap in performance caused by the increase in dimension which renders some of the heuristics such as #2 and #4 almost useless. This is likely a result of all the heuristics apart from heuristic #1 being heavily reliant on the observable information that is provided by the board to make a decision about which cell to uncover. So, as the board continues to grow and less information can be concluded from the starting position, it causes the agent to make worse decisions based on a faulty heuristic. As in the case of heuristics #2 and #4, if limited or misleading information can be extracted based on the neighborhoods of certain cells then it will render the heuristic ineffective. Similarly with calculating probability for heuristics #3 and #5, as less information is available on the board the usefulness of calculating probability decreases and the agent can become misled. To combat this, implementing heuristics which don't require excess information about the state to be effective is a possible solution. Similarly, supplementing the current heuristics with better functionality to handle the early cases of revealing cells when there isn't as much information such as through Monte Carlo Search, would be a way to minimize the decrease in accuracy. However, a balance must be maintained for the information given to the heuristic otherwise a similar issue as heuristic #5 could occur by combining multiple strategies. Adding too many strategies to a single heuristic can also result in time out or memory errors which was a major issue when testing the experiments for this project.

Despite the drastic decrease in performance across the heuristics for the various boards, the results illustrated the ability for a purely heuristic based agent to be a viable Minesweeper solver. Specifically, the sustained performance of heuristic #3 across the 3 boards highlights how these heuristics can be built upon to become effective and efficient solvers for Minesweeper boards of larger dimensions.

7 Conclusion

Throughout the paper, the problem of making effective and efficient solvers for the popular single player game Minesweeper was explored. Due to the stochastic nature of the game and logical patterns that can be exploited when playing the game, it has gained wide attention within the field of research in artificial intelligence. In this paper, work was done to create and test heuristic-based agents which could manipulate information from the game board to accurately determine which cells were safe to uncover. Five heuristics were formulated which consisted of a random move, neighborhood sum, minimizing mine probability, maximizing entropy of the distribution of the number of neighboring mines, and a combination of the previous 2 heuristics. The heuristics and the Minesweeper board were coded in Python and 40, 60 simulations of each heuristic were run on 3 modified Minesweeper boards with dimensions of 4x4, 5x5, and 6x6 with total number of mines of 2, 4, and 6 respectively. After running all the simulations, the results indicated that the best performing heuristic was minimizing mine probability which had win percentages of 92%, 58%, and 40% and average number of move of 3.27, 5.10, and 7.66 for the 3 boards respectively. The main trends extracted from the results were a positive correlation between the average number of moves and win percentage, a lack of correlation between heuristic complexity and accuracy, and a heavy dependency of the heuristic's performance on the Minesweeper board's dimensions.

To improve on the work done on this paper in the future, the main change would be to make the heuristics that were implemented compatible with the standard Minesweeper board dimensions. This would make it much easier to compare the findings of the experiments in this paper to previous work done in the field as they primarily use the beginner, intermediate, and expert board dimensions. It would also give better insight on the effects more mines has on the heuristics and whether they are still viable at much larger board dimensions. Similarly, measuring more advanced metrics such as total runtime and increasing the total amount of simulations for each heuristic and board combination to a sample size of 1,000 or 10,000 would improve the findings of the experiments. A larger sample size would assist in eliminating any abnormalities in the result and give a better representation of the true win percentage for each heuristic. Measuring a metric such as total runtime for each simulation would better indicate the viability of certain strategies and add another way to compare the efficiency of the various heuristics.

References

[1] David Becerra. *Algorithmic Approaches to Playing Minesweeper.* PhD thesis, Harvard College, 2015.

[2] Olivier Buffet, Chag-Shing Lee, Woanting Lin, and Olivier Teytaud. Optimistic heuristics for minesweeper. In *International Computer Symposium*, pages 199–207, 2021.

[3] Ade Candra, Mohammad Andri Budiman, and Rahmat Irfan Pohan. Application of a-star algorithm on pathfinding game. *Journal of Physics: Conference Series*, 1898(1):1–6, jun 2021.

[4] Evan Dieterle. Project4. Minesweeper project for CSCI 1933, University of Minnesota, Fall 2023.

[5] Shahar Golan. Minesweeper on graphs. In *Applied Mathematics and Computation*, 2011.

[6] Angela Tzujui Huang. *Develop heuristics to the popular Minesweeper game.* PhD thesis, California State University, San Bernardino, 2004.

[7] Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[8] Yinghao Li, Haorui Wang, and Chao Zhang. Assessing logical puzzle solving in large language models: Insights from a minesweeper case study. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, page 59–81. Association for Computational Linguistics, 2024.

[9] Emmet Nicholas. minesweeperonline.com. Website to play Minesweeper.

[10] Kasper Pedersen. The complexity of minesweeper and strategies for game playing. Technical report, University of Warwick, 2004.

[11] Pal Rujan. The minesweepers' baysian guide to survival. Technical report, Carl von Ossietzky University, 1998.

[12] Michèle Sebag and Olivier Teytaud. Combining myopic optimization and tree search: Application to minesweeper. In *Learning and Intelligent Optimization*, pages 222–236, 2012.

[13] Yash Sinha, Pranshu Malviya, and Rupaj Nayak. Fast constraint satisfaction problem and learning-based algorithm for solving minesweeper. Technical report, International Institute of Information Technology Bhubaneswar, India, 2021.

[14] Jinzheng Tu, Tianhong Li, Shiteng Chen, Chong Zu, and Zhaoquan Gu. Exploring efficient strategies for minesweeper. In *The AAAI-17 Workshop on What's Next for AI in Games? WS-17-15*, pages 999–1005, 2017.

[1]

[1]Anas wrote sections 3.1, 4.3, 5, 6, 7, and half of 1 and 3.3. Anas coded the neighborhood sum heuristic agent and experiments. Evan wrote sections 2, 3.2, 3.4, 4, and half of 1 and 3.3. Evan coded the minefield and other agents and also helped with experiments. Both authors worked together and contributed ideas.