Project 2: Minesweeper

Josh Cooper, Hoda Moustafa, Alex Wu
(joshcoop, section 2) (hmm96, section 7), (ajw183, section 2)

March 19th, 2020

Contents

Abstract

All group members affirm that all content included in this report and the supplied code are their own original work, which has not been plagiarized from online or another student's work from outside the specified group members. All graphs, figures, and data collected were generated only using the code provided.

This report is divided by the questions defined in the project description. Problem 1-3 pertain to the representation and inference for the advanced agent. We analyzed two different advanced agent versions (using rules of inferences (equations) and constraint satisfaction approach (CSP)), and then created a combined agent that incorporated both versions. Problems 4-5 analyze and compare the basic and advanced agent in terms of performance. The report is concluded with the two bonus parts and their respective analysis.

The sections relating to the Basic Agent and CSP Advanced Agent were contributed by Josh, Equations Advanced Agent by Hoda, and Combined Advanced Agent by Alex.

1 Problem 1 : Representation

For the beginning of the program, we decided on using a simplified ASCII representation of the board, storing the board display information as a list of list of chars, where "M" would represent a mine, "?" would represent a square that had not been uncovered by the agent, and a number would represent the clue given in the square. In the picture below, the first row of numbers above the dashes, and first column of numbers to the left of the vertical pipes represent row/column number info for easier location referencing by a human looking for errors in program logic.

```
    0 1 2 3 4 5 6 7 8 9
    ---------------------
0 | ? ? ? ? ? ? ? ? ? ?
1 | ? ? ? ? ? ? ? ? ? ?
2 | ? ? ? ? ? ? ? ? ? ?
3 | ? ? ? ? ? ? ? ? ? ?
4 | ? ? ? ? ? ? ? ? ? ?
5 | ? ? ? ? ? ? ? ? ? ?
6 | ? ? ? ? ? ? ? ? ? ?
7 | ? ? ? ? ? ? ? ? ? ?
8 | ? ? ? ? ? ? ? ? ? ?
9 | ? ? ? ? ? ? ? ? ? ?
```

```
    0 1 2 3 4 5 6 7 8 9
    ---------------------
0 | M 5 M M M 2 1 3 M M
1 | M M M 8 M 5 3 M M M
2 | 3 5 M M M M M 6 M M
3 | 3 M 6 M 4 4 M 6 M M
4 | M M M 4 3 4 4 M M M
5 | 4 M M 4 M M M 5 M 3
6 | M 5 5 M 5 M M 4 3 2
7 | M M 4 M 5 5 5 M 3 M
8 | M 7 M 5 M M M 4 5 M
9 | M M M 4 M M M 3 M M
```

Figure 1: The basic start of our board, with the information presented to the agent on the left, and the "answers" on the right

1.1 Board Representation

Given the desired dimension and mine density, our `make_board` function returns the covered board for the agent to solve, the solved board with everything revealed for reference, and the list of mine

locations on the board. As shown in Figure 1, the covered board on the left represents a new board with all locations unknown, and the reference board on the right shows the fully solved board.

1.2 Information Representation

For each cell that is revealed when chosen by the agent, if the cell passes a series of logic checks, said cells information is added to the knowledge base. For a cell to contribute a new fact to the knowledge base, the revealed cell must not be a mine, and must also have at least one unknown neighbor. Otherwise, that cell wouldn't be able to give us useful information, and would simply artificially inflate the knowledge base.

The knowledge base is constructed using a dictionary where the key is a tuple of the row and column location for the revealed cell, and the following information is stored as a list associated with that key:

1. Visual representation of cell; either a number from 0-9, "?", or "M"
2. Number of mines surrounding that cell
3. List of known mine locations which are neighbors to that cell
4. List of hidden locations which are neighbors to that cell
5. List of known safe locations which are neighbors to that cell

When we find definite information about a new cell, we update the knowledge base by changing at most eight facts, due to the idea that a new mine/safe location would only appear as a neighbor of at most the eight immediate surrounding cells.

1.3 Inferred Relationships Representation

1.3.1 CSP

For constraint satisfaction, we did not use any special relationship code, we just copied the state of the board and the knowledge base, and used an assumption on a single square to see how far out we could predict other spaces. We then collected all the new information gained from the assumption and took the intersection of the two sets of locations which came up as mined or safe (one set of safe locations via the assumption of safe on the initial square, and one set of safe locations from the assumption of the initial square being a mine, and the same information gathering technique for the locations of mines gained from the two differing assumptions.) We explain more about this in section 2.2 below.

1.3.2 Equations

For a given state of the board, we can represent the relationships between them using equations given 0 represents a safe cell and 1 represents a mine. For given cells in an equation, they are associated with a sum, n, which is the sum of how many remaining unknown mines there are. All equations are stored in a list where each equation is a list containing of `[list of unknown neighbors, number of remaining unknown mines]`. For example, if a cell has unknown neighbors A, B, C and 2 remaining mines with unknown locations, the equation would be modeled as : `A+B+C = 2`, where each letter from A, B, C is a coordinate point on the board, and the value 2 shows that among the three cells, 2 of them are mines and 1 of them is safe. This representation is used in the rules of inferences approach.

An example of this representation is given the following board state below, we could generate an equation of $[[0, 1], [1, 0], [1, 1]], 1]$. The list of unknown neighbors that haven't been discovered are $[0, 1], [1, 0], [1, 1]$, and the 1 indicates there is 1 mine left to be discovered. If we were to put this in equation form, it would be represented as : `[0,1] + [1,0] + [1,1] = 1`

Figure 2: Sample 5x5 Board with [0,0] cell revealed

The way these equations are manipulated and interconnected with each other are mention in section 2.3 below.

2 Problem 2: Inference and Decisions

2.1 Basic Agent

The basic agent follows the pseudocode provided in the project pdf. It only looks at a single clue, and uses a comparison between the number of unknown neighbors of a location and the number of mines around a location (both known mine locations and total possible mines) to either conclusively mark a new location as a mine or as safe. In the code we refer to these as Rule 1 and Rule 2, where

Algorithm 1: Rule 1: Unknowns are all mines

```
num_mines = the locations clue (ie how many mines are adjacent to it)
known_mines = how many neighbors are already identified as mines
hidden_neighbors = the locations of all neighbors which are still unknown
if num_mines - length(known_mines) == length(hidden_neighbors):
    all hidden neighbors are marked as mines
```

Algorithm 2: Rule 2: Unknowns are all safe to be revealed

```
num_mines = the locations clue (ie how many mines are adjacent to it)
known_mines = how many neighbors are already identified as mines
hidden_neighbors = the locations of all neighbors which are still unknown
if num_mines == length(known_mines):
    all hidden neighbors are marked as safe
```

When either of these rules identifies a new location as safe/mined, we update the knowledge base and rescan it again to see if this new information allows any further single clue inferences. If we reach the end of the knowledge base without finding out any new information, we generate a pair of random numbers, and flip over that location on the board, add that new locations fact to the knowledge base, and restart the loop of attempting to see if the new fact leads to any new information

2.1.1 Decisions

The basic agent will always attempt to do a full scan of all facts present in the knowledge base, to see if any new information can be gleamed from single facts by using Rule 1 or Rule 2. If a full scan is made through the knowledge base, and no new information has been found, the agent resort to choosing a random cell as next move. The agent randomly generates a pair of numbers, and if that location is a cell is currently unknowns, then proceeds to pick that cell, add the new fact based on that cell to the knowledge base, update the knowledge base as needed based on this single cell, and then attempt another full scan of the knowledge base.

2.2 Constraint Satisfaction approach (CSP):

The COWAN DOES IT BETTER Algorithm:

Constraining Our Wily Adjacent Neighbors During Onerous Exploration Steps Immediately Thereafter Building Equations Through Trial and Error Reductions

To improve performance above the level of the basic agent, we first implemented a version of constraint satisfaction with contradiction checking, to see if we could make assumptions about an un-

known location on the board which might lead to new definite information elsewhere on the board. Our CSP approach works in two distinct stages. First, it uses the basic agent logic to explore as much of the board as possible by single clues. When it has exhausted all those possibilities, it proceeds to create a frequency dictionary of how often all the unknown locations currently present in the knowledge base show up (ie, if location (3,5) shows up in three different facts, the frequency dictionary would contain (3,5) -> 3). Once that unknown frequency dictionary is created, every location which shows up in at least 2 facts is extracted and used as the list of possible locations to execute a Constraint Satisfaction exploration on.

Starting with whatever location shows up in the most number of facts in the knowledge base, the agent makes two parallel assumptions, checking what would happen if this location was either a mine, or a safe location. On both tracks, once the initial location has its value assumed, we switch back to using basic agent logic for as long as we can. Once both "assume mine" and "assume safe" tracks have run as far as they can using only basic logic, we compare the resulting boards to see if any spots have come up with the same answer (has a location been definitively marked as safe/mined in both the "assume safe" and the "assume mine" tracks.) For every location which produced the same definite fact in both tracks, we update the main board, and the main knowledge base, and restart our basic agent with these new facts.

There are two other possible actions which can occur during this CSP approach. First, if the most frequent unknown square doesn't actually produce any locations which give definite "safe/mined" information from the parallel assumption tracks, we choose the next most frequent unknown, and proceed through the algorithm again. If we run through all of our unknowns and haven't found any new locations to change, this stops our advanced agent, and brings us out of our main loop to choose a new random spot on the board again.

The other possibility is that on one of our assumption tracks, our initial assumption causes a contradiction somewhere down the line. Say we assume location (3,5) is a mine to start off the CSP "assume mine" track. If this assumption causes another fact to come up with a statement which says that there must be an addional 2 mines around another location, but that location only has one unknown space left to possibly place mines, this is a contradiction caused by our initial assumption. When a situation like this occurs, we can immediately go back and mark the first location (3,5 in this example) as a safe space, due to our initial assumption that it was mined created an impossible situation down the line.

We show a graph of CSP effectiveness in section 2.4 when we combined this approach with our next approach to make a super advanced agent.

The whole CSP algorithm ends up looking like this. (Algorithm on next page)

Algorithm 3: assume_a_location($agents\_board, reference\_board$)

```
while no changes have been made on agent_board:
    fact_dict = create from agents_board //knowledge base
    frequent_unknowns = grab info from fact_dict
    for a location in freq_unknowns:
        assume location is mined
        new_info_1 = any locations which can be definitively proven using basic agent
        no_mine_contradiction = has initial assumption caused problem?
        if no_mine_contradiction:
            assume location is safe
            new_info_2 = any locations which can be definitively proven using basic agent
            no_safe_contradiction = has initial assumption caused problem?
        if not no_mine_contradiction:
            mark location as mine on agents_board
        else if no_mine_contradiction and not no_safe_contradiction:
            mark location as safe on agents_board
        else:
            find intersection of new_info_1 and new_info_2
            mark locations on board as safe/mined, depending on info from assumption tracks
    if changes have been made, repeat while loop
return agent_board
```

2.2.1 Decisions

The major choice made in this version of the CSP algorithm was how to choose the limitations on the frequent unknowns list. We selected all unknowns appearing in two or more facts after running simulations on multiple values, and finding that accuracy didn't increase when selecting all unknowns across all facts, but the average solving time increased by a factor of two or more depending on board size. Interestingly, when we attempted to limit the choice of the frequent unknown to all unknowns which appeared in 3 or more facts, the overall accuracy of our solution went down across all mine densities, due to the fact that it had to make more random choices because the CSP was running out of new facts sooner. We've included a graph below showing the accuracy changes and time gains across these three different values of the frequency limits. "Advanced Freq 1" is the version where our CSP uses unknowns which appear in at least two facts, while "Advanced Freq 0" is every unknown is attempted, and "Advanced Freq 2" is the version which only uses unknowns which occur in three or

more facts in the knowledge base. We also show the basic agent, which is always beaten in terms of accuracy by any of the advanced agents even when they're only using CSP, and not any of the other methodologies we implemented.

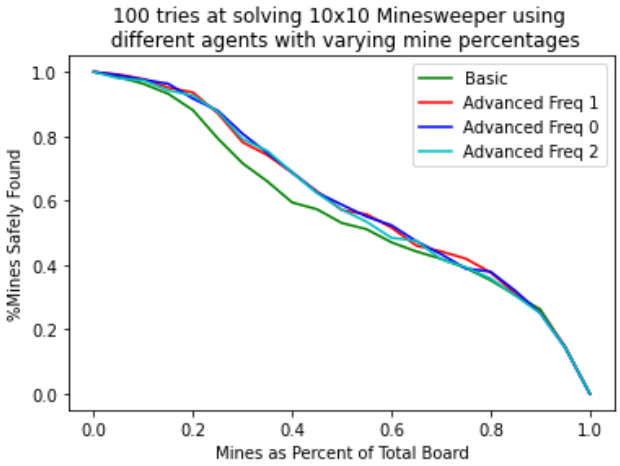

Figure 3: Total Score with different freq values

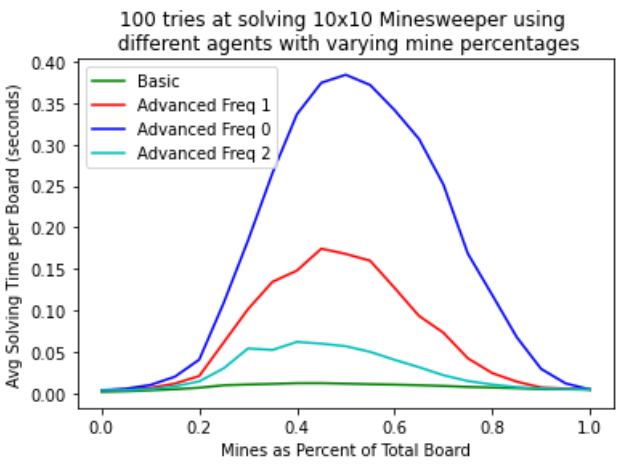

Figure 4: Average Time to Solve

2.3 Rules of Inference approach

The ARAVIND Algorithm:

Advanced Reconnaissance Agent Verifying Intersecting and Numerous Data

Another method for advanced inferring is creating equations from the information given on a current state of the board. Each equation can be thought of as a set, with a corresponding number of unknown values. As equations are being made and updated, the algorithm keeps track of the following:

1. Known Mines (KM): a set of points inferred from the equations that are confirmed to be mines
2. Known Safes (KS): a set of points inferred from the equations that are confirmed to be safes
3. Equations made: a list of possible equations that are produced using the given state of the board

If anything can be established using the Basic Agent methods, then those changes are made first and reflected on the board before creating equations. The algorithm first starts by creating equations given the knowledge base of the board at a certain state. The following algorithm shows the pseudocode for how the equations are made:

Algorithm 4: apply_equations($agents\_board, reference\_board$)

```
1 fact_dict = create from agents_board //knowledge base
2 equations_before = grab info from fact_dict
3 equations = []
4 for each 2 equations in equations_before:
5    convert eq1 and eq2 to sets
6    if eq1 is a subset of eq2
7        final_eq = eq2 − eq1
8        subtract the number of unknown mines for each eq from each other
9        add final_eq to equations
10   else if eq2 is a subset of eq1
11       final_eq = eq1 − eq2
12       subtract the number of unknown mines for each eq from each other
13       add final_eq to equations
14 return(equations )
```

Given an equation 1 with `[set of cells1, count1]` and an equation 2 with `[set of cells2, count2]`, we can construct a new equation where `[set2 - set1, count2 - count1]` given that one is a subset of the other. If neither are subsets of each other, then a second equation can't be formed as we will not necessarily be getting correct inferences. The equation resulting from 2 different equations is a result of one being a subset of the other. The reason we create equations using the subset condition is because we can guarantee the resulting equation we create will almost always have something to infer from. If we simply took the difference between 2 equations, the agent may incorrectly infer cells as mines or safes if there isn't enough information. However, if an equation is a subset of another, information may be inferred from the remaining equation after combining them, depending on the total number of remaining mine numbers. For example, if we have the 1st equation as $[\{A, B, C, D, E\}, 4]$ and the 2nd equation as $[\{D, E\}, 1]$, we can combine them to reveal $[\{A, B, C\}, 3]$, which provides new information we previously didn't know. The inference process of analyzing the equations follows the pseudocode below:

Algorithm 5: infer_equations($agents\_board, reference\_board, equation, KS, KM$)

```
changed = False //True if can infer from equation
eq = [set of cells, n] //n = number of unknown mines
if n == 0: //all set of cells are safe
    mark cells from equation on the board as safe
    add cells to known_safes set
    changed = True
if n == length of set of cells: // all set of cells are mines
    mark cells from equation on the board as mines
    add cells to known_mines set
    changed = True
return( changed, agents_board, known_safes, known_mines )
```

Let n be the number of unknown mine locations. This number is relies on the basis that a mine cell has a value of 1 and a safe cell has a value of 0. For example, if we have $[\{A, B, C\}, 1]$, then we know from the equation that one of cells A, B, and C are a mine and the other two are safe. We can then make inferences using this representation of 1 for a mine, 0 for safe. For each equation given, the inference results relies on 2 cases.

1. Case 1 (n = 0): if n is 0, then we can infer all the cells in the given equation are safe locations.
2. Case 2(len(equation) = n): if the number of cells in the equation is equal to n, then we can infer that all the cells in the equation are mine locations.

In Case 1, if the given equation is $[\{A, B, C\}, 0]$, then we can infer that A,B, and C are all safe cells. Likewise, for Case 2 if the given equation is $[\{A, B, C\}, 3]$, then we can infer that A,B, and C are all mine cells. Once a cell is determined to be safe/mine, the board is then updated with the new knowledge and returned. When an inference is made and the board is updated, we then update the other equations to see if the newly revealed equation appears in any other equations made initially. The update algo is based on the following pseudocode:

Algorithm 6: update_equations($equations, KB, agents\_board, reference\_board, KS, KM$)

// KB = knowledge base

1 for each $[eq, n]$ in equations:

2 if eq cell in ks: cell is safe

3 remove cell from equation

4 if eq cell in km: cell is mine

5 remove cell from equation

6 n -=1 // decrement unknown mine count

7 return($equations$)

The algorithm goes through the equations and checks if any of the cells present in the equations are also a known safe or a known mine. If the cell is a known safe, then it is simply removed from the equation. If it's a known mine, then it's removed from the equation and n is decremented. We decrement n since we figured out one of the mines of the equation, and update accordingly. Once the equations are updated, we then check to see if we can infer anything more, and if not the agent picks a random cell as the next move after inferring everything possible from the equations given.

The current state of knowledge is updated after a clue is discovered by rebuilding the knowledge base to reflect the changes of the discovered cell(s) after uncovering them from a "?" to their true value. Before resorting to picking a random cell, the program would attempt to solve the equations for any new information, and mark it on the board as it goes. Once everything has been deduced and the cells have been revealed, the agent then moves on to picking a random cell. We can be sure that the program deduces everything for a given clue by using the subset method rather than simply removing the intersections between 2 equations. Without using subsets, incorrect info may be inferred. For example, if given the 2 sets of equations $[\{A, B, C\}, 1]$ and $[\{B, D, E\}, 1]$, and we take the difference between them we end up with the final equation as $[\{A, C, D, E\}, 0]$. According to our rules of inference, this is Case 1 where all the cells are safe since n = 0, but this may not be the case as there isn't any other info we can use to verify. Therefore, by checking for subsets, we can guarantee that the program deduces everything from a given clue before moving on. In addition, we can visualize the equations that are created and check them against the board the equations are based on.

2.3.1 Decisions

First, we try to use the basic agent and see if we can deduce anything given the single clue. After attempting to use the basic agent, whether successful or not, the program then proceeds to test the equations, and updates the board whenever new information is found. The program attempts to

uncover everything possible during the current step before picking a random cell. During the next round when a random cell is picked, the board is updated with the new inferences, if any. When nothing more can be inferred from the basic agent and subsets, a cell is picked randomly.

2.4 Combined Approach

The CHARLES Algorithm: Combined Heuristics Agent Resolving Last Exotic Squares

We wanted to combine both the CSP approach and the Rules of Inference approach into one to see if any improvements could be found from having the two work alongside each other. As shown in Figures 5 and 6, there were cases in which CSP approach outperformed the Rules of Inference approach and vice versa. By combining both, we hoped that the number of random guesses would be the minimum of both methods and the number of bombs safely found would be the maximum of both. As shown by Figures 7 and 8, their performance were considerably close to one another, which led us to combining both approaches together in the hopes that the overall performance would be greater than the individual algorithms.

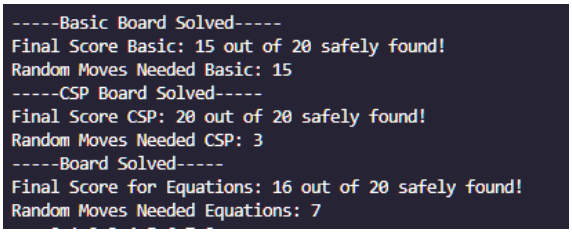

Figure 5: Scenario where the CSP approach had the advantage

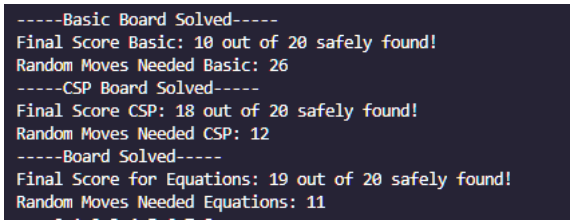

Figure 6: Scenario where the Rules of Inference approach had the advantage

The algorithm works by first going through the board as much as it can using the basic agent, similar to the other methods, and then pivoting over to the Rules of Inference approach. Once a scenario is reached where both the basic agent and the Rules of Inference approach cannot obtain conclusive new information, the CSP approach is applied. This cycle of swapping through the basic agent and the two methods is repeated until no more squares can safely be opened. At that point, the agent then falls back on selecting a square at random as a final resort. The algorithm is modeled by the following pseudocode:

Algorithm 7: combined_agent(agents_board, reference_board, number_of_mines)

```
fact_dict =  //knowledge base
score = number_of_mines //only for keeping track of the agent's score.
number_of_guesses = 0
find unknown squares on the board
while there are still unknown squares on the board
    uncover a random square on the board
    number_of_guesses += 1
    if the random square we explored was a mine
        subtract one from the score
    attempt to build a fact dictionary from the explored squares on the board
    apply basic agent logic to the dictionary
    if unknowns == 0: //the basic agent has cleared the board
        the board has already been solved
        while true:
            apply the rules of inference method on the board
        if the board was changed from the rules of inference approach:
            find the unknowns still on the board
            continue //don't continue onto the csp agent yet
        if unknowns == 0: //the basic agent has cleared the board
            the board has already been solved //prevent the algorithm from applying csp
        while true:
            apply the csp method on the board
    find the unknowns still on the board
return(score, number_of_guesses)
```

In choosing which of the two approaches to run first, we based the decision mostly on runtime, as there was very little difference in both methods for their performance in both success rate of finding mines and number of random guesses, as seen by figures 7 and 8 (Rules of Inference approach represented by "Equations"). However, as shown in figure 9, the Rules of Inference method consistently ran substantially faster than the CSP method. As a result, to minimize the number of nodes the slower method checks and thereby reduce the amount of time the advanced agent takes to solve the board, we went with applying the Rules of Inference approach first, and then turning to the CSP approach when we cannot deduce more information.

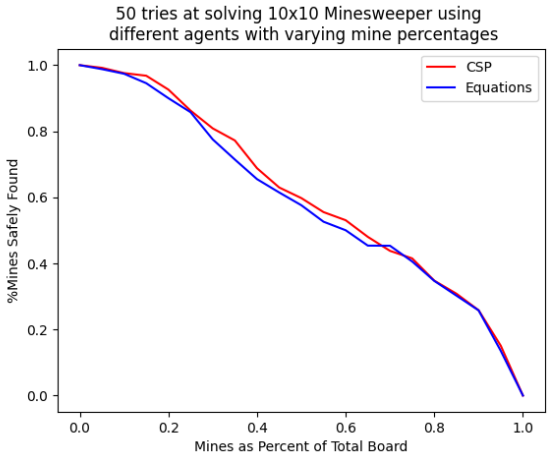

Figure 7: Rate of success for CSP vs Inference

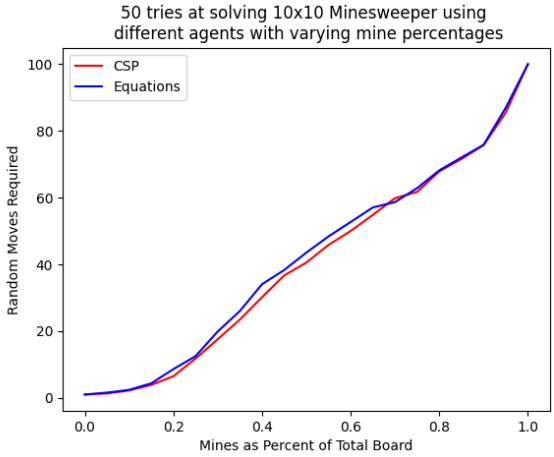

Figure 8: Number of random choices for CSP vs Inference

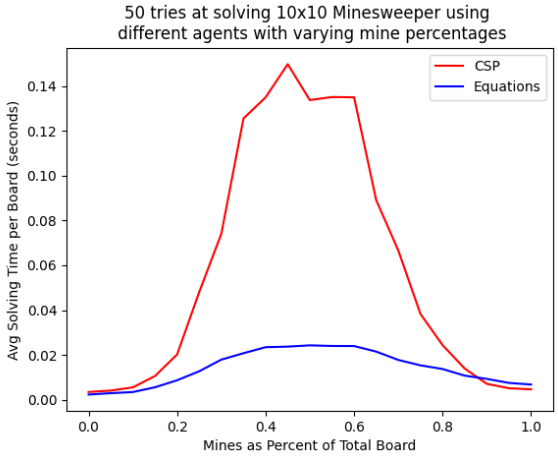

Figure 9: Runtime of CSP agent vs Rules of Inference agent

2.4.1 Decisions

Due to the way the algorithm is designed, much of the emphasis for the inferences past the basic agent logic will be done by the Rules of Inference agent. Once the basic agent stage has concluded, the agent moves to testing equations and updating the board where applicable. Only when the Rules of Inference agent has not created a change in the state of the board is when the agent will move towards a CSP approach, creating a frequency dictionary and exploring possibilities for the remaining squares. As such, the majority of work the CSP agent performs on the board for the combined approach will be at the very end on the final unknown squares of the board.

3 Problem 3: Performance

3.1 Advanced Agent CSP

In section 2.2.1, we discussed a few of the simplifying choices we used during the CSP approach, and the differences between accuracy and average time complexity. This is primarily due to the fact that no matter the point in the specific CSP evaluation, the need for space in maintaining the knowledge base is always at minimum 3 times as much as the regular board (main knowledge base and the two distinct assumption bases). However, despite the increase in space/time complexity, our CSP was able to make valuable inferences which contributed to an overall increase in average score.

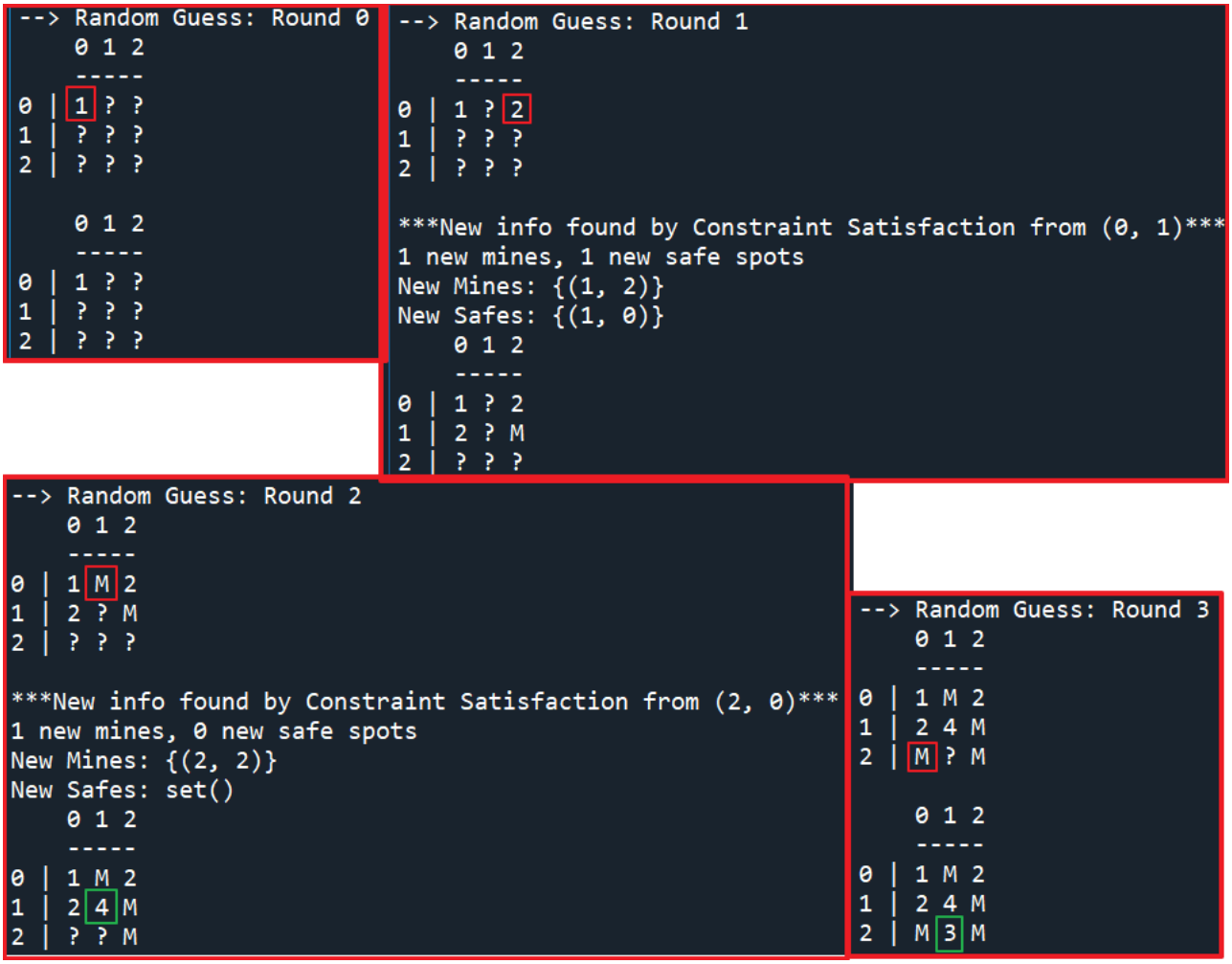

Figure 10: Here is a small example of the CSP algorithm detecting new locations which would be found only after repeated random location selection by the basic agent. Green border cells indicate locations marked by the basic agent, and red border cells indicate a location chosen by random choice. All indicated locations are in the form (row, column)

3.2 Advanced Agent Rules

We used a 10x10 board with 20 mines to include the play-by-play progress of the advanced agent using the rules of inferences (equations) to solve the board. The cells that are uncovered through the basic agent inferences are omitted from the steps below to highlight the information the equations uncover. The cell indicated by a white square is a random guess by the basic agent. Each cell indicated by a red square is a new clue discovered using equations. The number in parenthesis indicates the number of random guesses the agent needed to take. The solved board is shown at the end with a yellow square indicating the mine that was revealed.

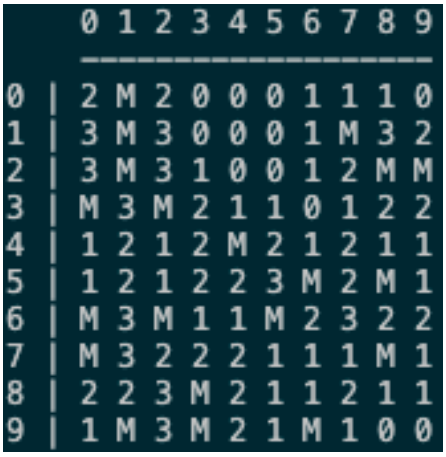

Figure 11: Reference Board Agent will Solve

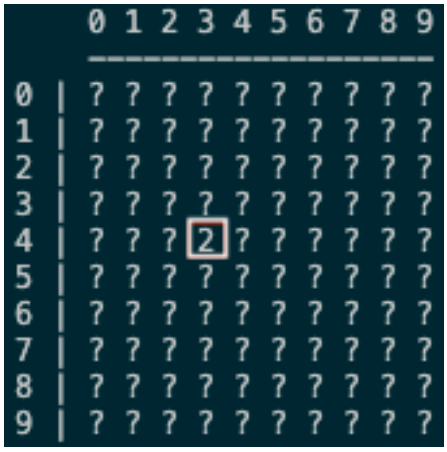

Figure 12: Initial start move

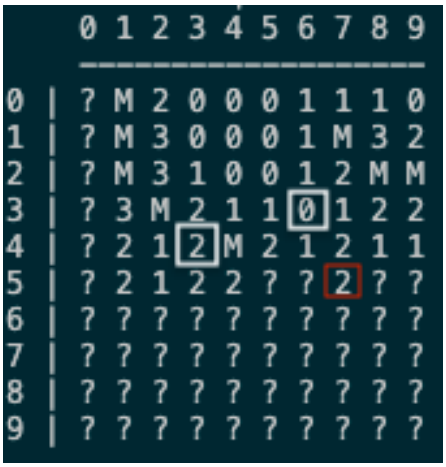

Figure 13: $[5, 7]$ inferred (1)

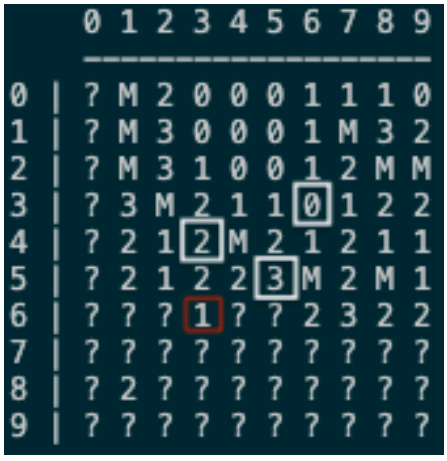

Figure 14: $[6, 3]$ inferred (2)

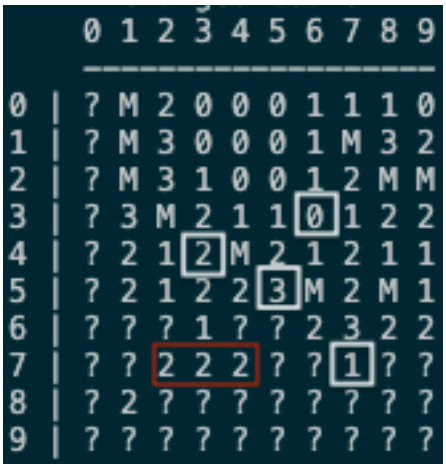

Figure 15: $[7, 2], [7, 3], [7, 4]$ inferred (3)

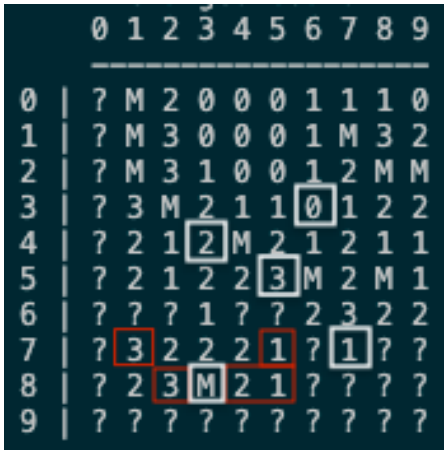

Figure 16: $[7, 1], [7, 5], [8, 2], [8, 4], [8, 5]$ inferred (4)

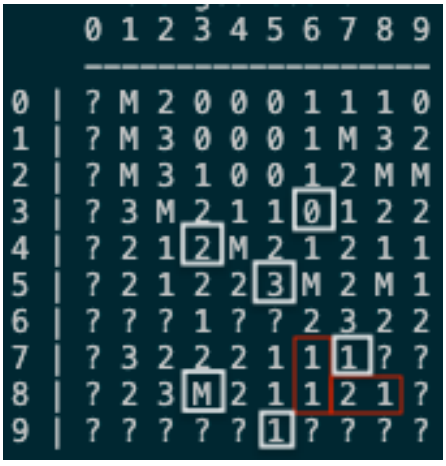

Figure 17: $[7, 6], [8, 6], [8, 7], [8, 8]$ inferred (5)

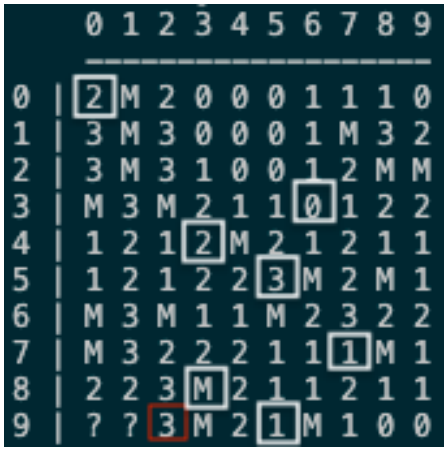

Figure 18: $[9, 2]$ inferred (6)

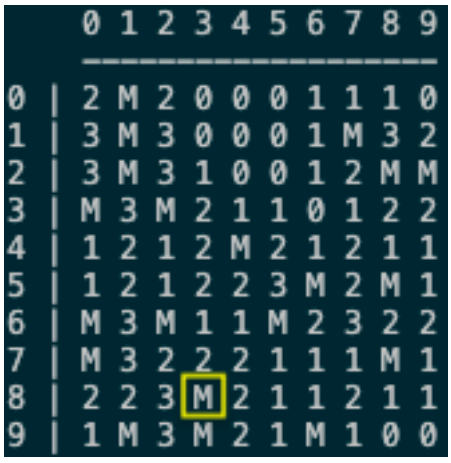

Figure 19: Completed board (7)

After successfully completing the play through, the agent managed to find 19/20 mines safely, with 1 mine randomly chosen during the basic agent choices (shown in the yellow square). The

decisions the agent took to progress were similar to those of a human playing the game. The only part I wouldn't agree with is when the agent resorts to choosing a random cell and ends up choosing a mine (in Figure 16). I would've chosen the next random cell closer to the cells revealed that have a lower number. For example, picking an unknown cell around a cell marked by "1" has a lower chance of getting a mine than a cell shared by several surrounding cells of value "2". The decisions that surprised me were those of the basic agent in relation to the new clues found. Despite the basic agent being inefficient alone, when grouped with more advanced analysis, the number of mines safely found increases and there are fewer random guesses. For this particular play-by-play game, the basic agent took 11 random moves with 18/20 mines safely found. Although the random moves aren't exactly the same and the difference between the basic and this advanced agent version aren't vastly different, it just shows how the extra clues found through the equations allowed the agent to solve the board in fewer moves and more safely.

3.3 Advanced Agent Combined

We used a 10x10 board with 20 mines to show the play-by-play progress of the combined agent to solve the board. Again, we will omit progression made solely by the basic agent to focus on inferences made by CSP and Equations. Cells indicated by white were made via random guessing. Cells indicated by red were made with the Rules of Inference approach. Cells indicated by blue were made with the CSP approach. We will use the same notation as the previous example; so the number of cumulative random guesses (including the first move) will be in parentheses, and the specific moves inferred at a given point in time will be listed. Squares marked by the Rules of Inferences agent will be accompanied by the phrase "inferred by Eq". The same goes for CSP, which will similarly be accompanied by "inferred by CSP".

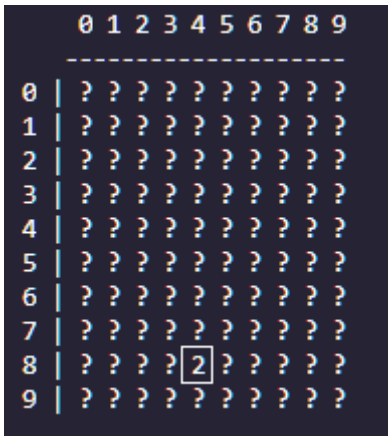

Figure 20: Initial board and starting move (1)

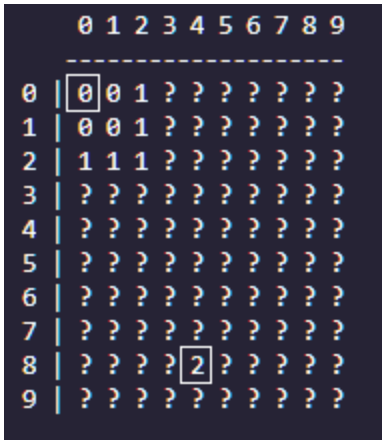

Figure 21: 2nd random move expanded upon by basic agent (2)

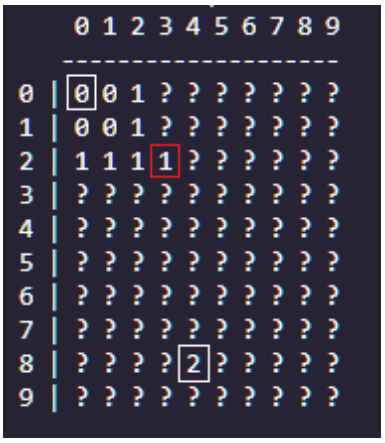

Figure 22: $[2, 3]$ inferred by Eq

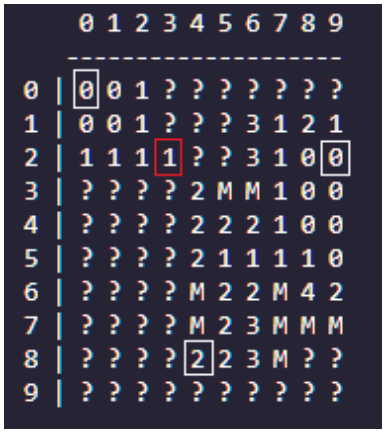

Figure 23: 3rd random move expanded upon by basic agent (3)

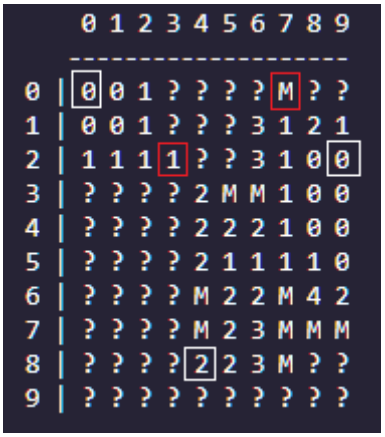

Figure 24: $[0, 7]$ inferred by Eq

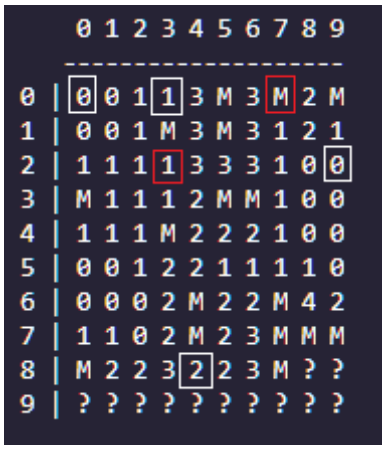

Figure 25: 4th random move expanded upon by basic agent (4)

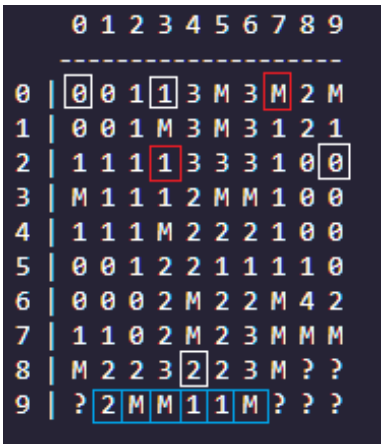

Figure 26: $[9, 1-6]$ inferred by CSP

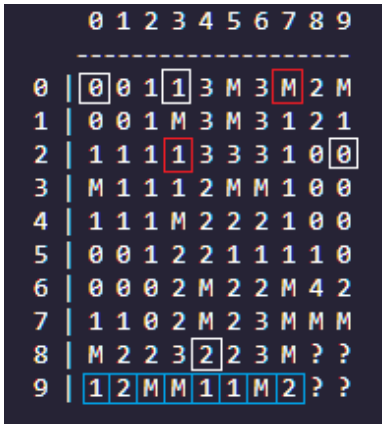

Figure 27: $[9, 0], [9, 7]$ inferred by CSP

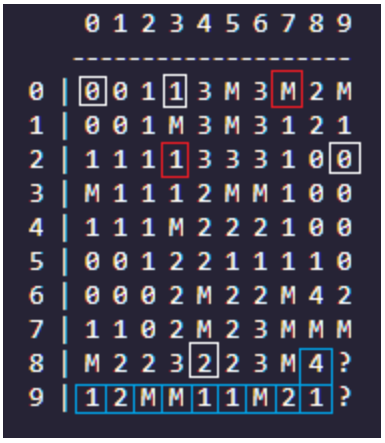

Figure 28: $[8, 8], [9, 8]$ inferred by CSP

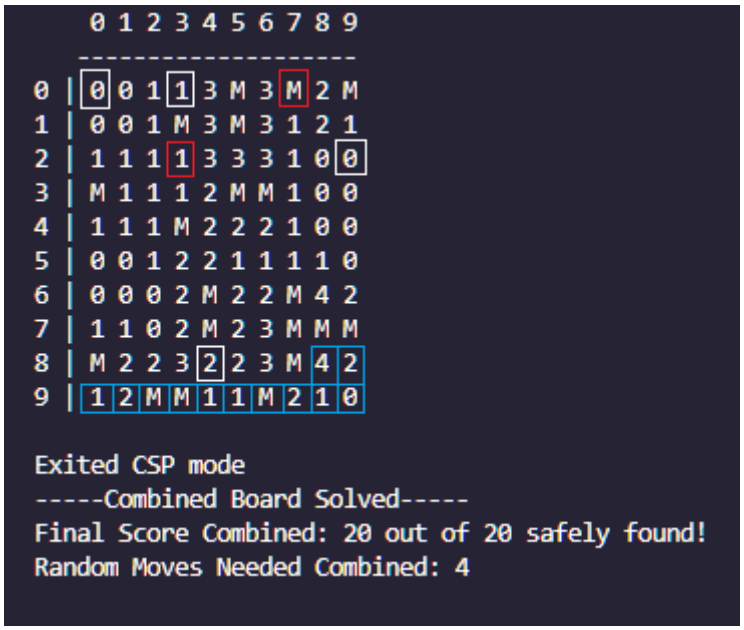

Figure 29: $[8, 9], [9, 9]$ inferred by CSP. Board completed

On this particular board, the agent successfully located every bomb on the board without detonating any with only 4 random moves (including the initial move). As expected, the agent ran through with getting as much information from equations first. At figure 25, the Rules of Inference agent ran into a dead end, where it stated that no more unknown squares could be uncovered using the equations it had in the knowledge base. On the solo Rules of Inference agent, this would have resulted in another random move, and with 3 mines out of the 12 uncovered squares, did not represent the best of odds. The agent pivots over to CSP mode, where it first uncovers $[9, 2]$ and $[9, 3]$ as mines via contradictions with squares $[8, 1]$, $[8, 2]$, and $[8, 3]$. The surrounding squares are then cleared, with another mine being uncovered at $[9, 6]$. From there, the agent carefully uncovers the remaining unknown squares and exits CSP mode, thereby completing the process and the board.

4 Problem 4: Performance Analysis

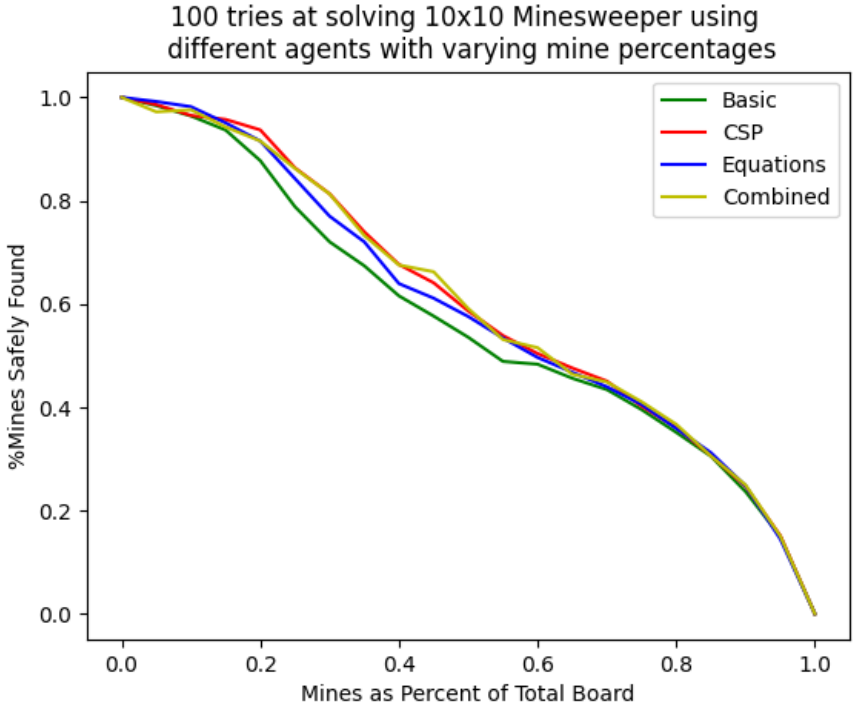

Figure 30: 10x10 Board With Average final score for Basic agent compared to Advanced Agent Versions (equations, CSP, and equations+CSP together)

According to the graph above, the results agree with our intuition. Overall, the advanced agents perform notably better than the basic agent. Specifically, between mine density percents of 0.1 to 0.7 is when the combined advanced agent successfully flags mines more often than the basic agent does. The simple and advanced algorithms perform the same when the mine density is very low and very high. With those mine densities, the basic agent can uncover more information from a single clue than when the mine density is more equally spread out (more squares with "0" values or "M" values). The basic agent however never beats the advanced. The advanced agent is more frequently able to infer more information than the basic agent when it takes less random moves to solve the board.

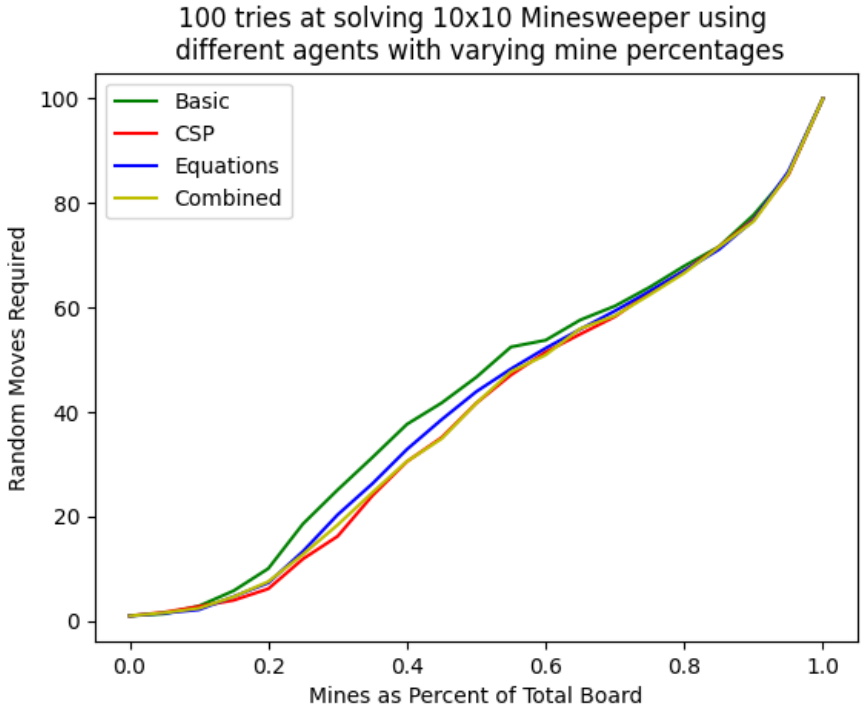

Figure 31: 10x10 Board With Number of Random Moves Required for Basic agent and Advanced agents (equations, CSP, and equations+CSP together)

As seen in the figure above, the advanced combined agent takes less random moves to solve the board between 0.1 and 0.7 mine density percents, which is concurrent with the results from the previous graph when the advanced agent performs better than the basic. Since the advanced agent chooses less random cells than the basic agent, it therefore infers more from the equations and CSP than the basic agent does from single cells.

We decided to graph the different results of each advanced agent version to compare their progress. After graphing the combined advanced agent, we noticed the performance is a little over the individual advanced agents performance. We expected the performance to be significantly over the individual agents, however it hovers around the equations and CSP versions, which is account for the random selections for the starter moves and the basic agent random choices. Between the equations and CSP versions, we notice that CSP performs a little better overall, which is reflected in the combined version. The offset of the CSP version however is that it's more time consuming than the equations version, as the algorithm essentially tests each cell twice with 2 different versions of the knowledge base.

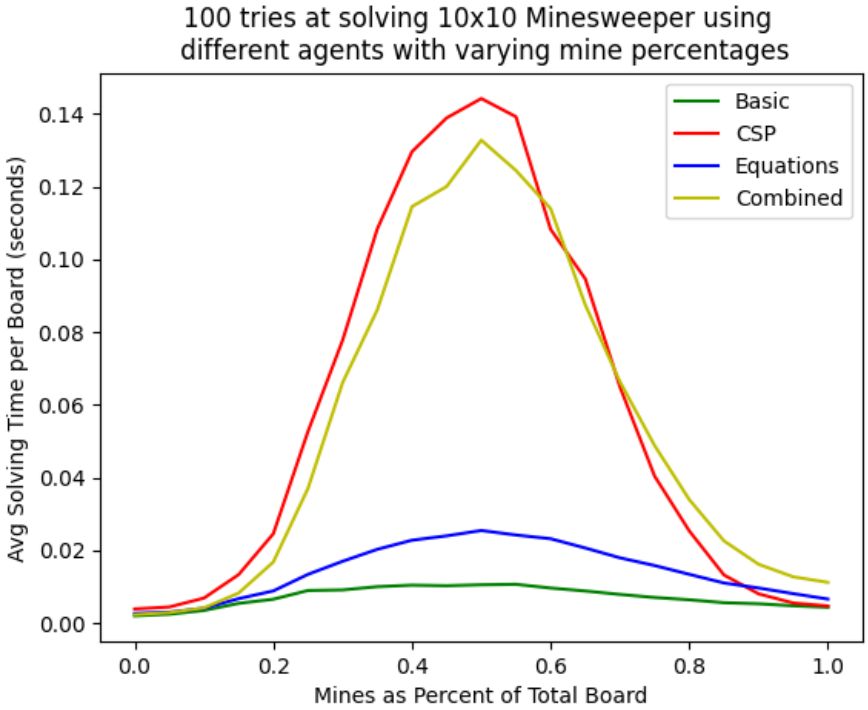

Figure 32: 10x10 Board Time Required for Basic agent and Advanced agents (equations, CSP, and equations+CSP together)

As shown in the figure above, the CSP advanced agent takes the longest time to complete the board, while the equations advanced agent finishes as close to the basic agent's time. The final combined advanced agent version hovers below the CSP agent time and above the equations agent time. This is expected, as the final version of our advanced agent is essentially running two different kinds of checks to discover new clues.

Minesweeper becomes hard when the agent chooses more mine cells as the next move than safe cells. This happens as the mine density increases. As seen from the two graphs above, as the number of mines successfully found decreases, the number of random moves needed (less inferences) decreases. This agrees with our intuition, if there are more mines, the chances of the agent randomly picking a safe cell are slim, and thus the knowledge base doesn't gain as much knowledge that would help in inferring, forcing it to pick more random cells.

5 Problem 5: Efficiency

As mentioned in the performance analysis above, the combined advanced agent takes a longer time to solve the board compared to the equations or CSP methods individually. This is expected, however it may be complicated when the board sizes increase, and thus poses an issue. As shown in Figure 31, despite the basic agent taking the fewest time to compute the boards, the amount of random moves needed are consistently higher than the CSP agent and combined agent. This is a sacrifice we had for our final agent, where it's average solving time is notably slower than the basic agents, but the overall success is higher and the random moves needed are lower.

The large difference in runtime between CSP/combined and the Rules of Inference approach, that can be attributed to the assumption step, in which at every square that needed to be checked, two tests were performed in parallel, as previously mentioned in the CSP section in problem 2. We checked to see if marking a square as safe or flagging it as a mine would lead to new information. To accommodate for these simulations, a copy of the fact dictionary was provided for each, thereby dramatically increasing the associated overhead for the overall agent. However, this can be considered a problem specific constraint, as the two extra fact dictionaries provide a foundation for the strategy, and is not something that can easily be replaced.

6 Bonus 1: Global Information

For our version of global information, we took the very small step of creating a new fact in our knowledge base which contained the location of all known mines, known safes, current unknowns, and the total number of mines on the board. The fact format matches the form we used in the basic agent and advanced agent. This large fact was constantly being updated every time a inference was made by any of our basic/equations/CSP/contradiction steps, and whenever a random spot was selected. By design, we added this fact in the front of the knowledge base, so that when our other agent techniques iterate through the knowledge base, it always checks the global fact first. This allows a possible calculation time cutoff if the global fact ends up showing that the number of unknowns matches the number of mines left to find, or if the number of mines left to find is 0. This fact added a decent amount of overhead in terms of calculation time, and didn't have much of an effect until the mine density on the board was more than 60 percent. Once the mine density was larger however, this added clue contributed to a decent increase in overall score (how many mines are safely identified). There's also the added benefit of instant solving any board with 0 mines or 100 percent mine coverage by making the global fact be looked at first, but that's just a cool outlier, not any generally useful trend. In terms of overall efficiency/tradeoffs for use, global info probably could have been implemented slightly differently by us in our fact dictionary, but given that it was tacked on at the end of our project, we didn't want to redesign the whole codebase for this bonus section.

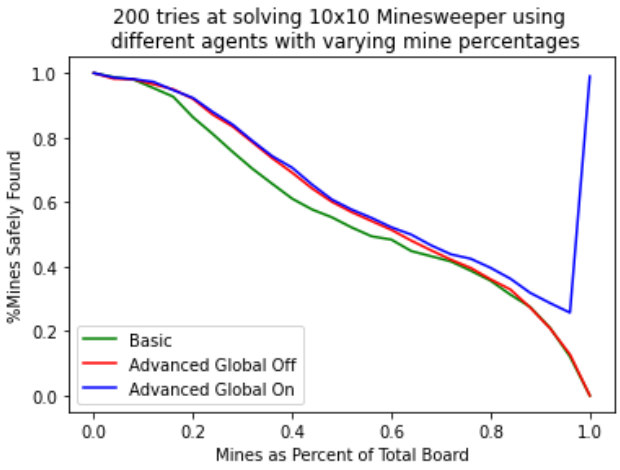

Figure 33: Agent Score Comparisons

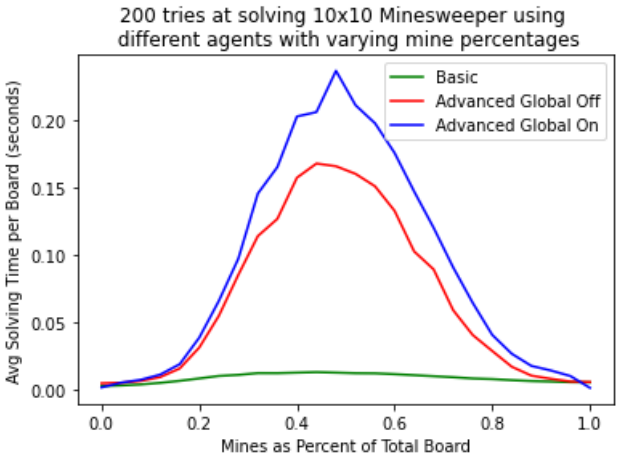

Figure 34: Agent Avg Time to Solve Comparisons

7 Bonus 2: Better Decisions

To make our agent choose better positions when it ran out of inferences via the basic/advanced strategies, we attempted to take an estimate of how likely a specific unknown cell might be a mine. For the purposes of this example, we are going to reference the figure below and explain the steps taken by our move selector.

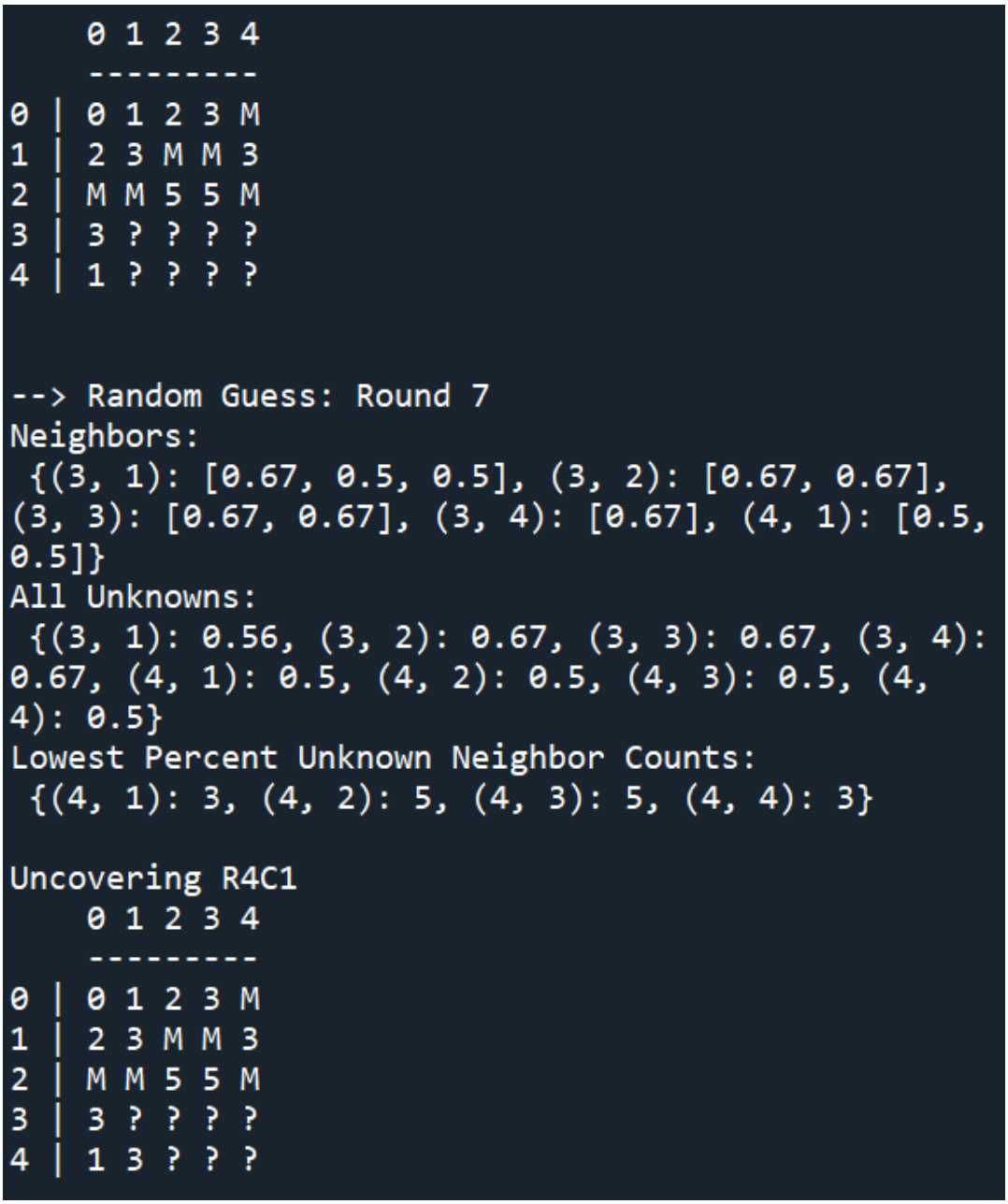

Figure 35: Showcasing all the aspects of the advanced selector

Given the board state shows at the top of the image, our selector first identifies all current unknown cells which are neighboring some other known cell. We will start our example by trying to

figure out the chance that location (3,1) will be mined (This location, like all locations in the picture and the description, are in the form of (row#, column#). This unknown cell is a neighbor of 3 other cells which have clues (3,0, 4,0, and 2,2) Each of those three revealed clues is looked at individually to help figure out the chance that any of their neighbors might be a mine. Lets look at location (3,0) first. It has two revealed mine neighbors, two unknown neighbors, and a total mine clue of 3. That means that there are 2 unknown spots with only one mine left to place. Naively, this single clue means that those two unknown locations (relative to 3,0) would each have a 50 percent chance of being a mine, and therefore our unknown in question (3,1) would have a mine chance of 0.5. When we apply the same type of naive logic to the single clues gained from (4,0) and (2,2), we find the chance of location (3,1) being a mine due to those single clues would be 0.5 and 0.67 respectively. Once we do this procedure for all unknowns relative to a known clue, we average the probabilities on any location to get one single chance. On our (3,1) spot, we would be averaging the 0.5, 0.5, and 0.67 chances from the three clues next to (3,1) to give that location a final mine chance of 0.56. In the absence of any clue, we assume that unknown location has a 0.5 chance of being a mine.

Then, when we have completed the mine chances for all the unknowns left on the board (the All Unknowns line in the figure) we select the locations which have the smallest chance of being a mine. In the case of the figure above, multiple locations have the same minimal chance of 0.5. If a situation like this arises, we collect all these locations, and find out how many unknown neighbors each of these new locations has. We theorized that the best choice would be a location which has the smallest number of unknown neighbors, as it would be the one most likely to produce an easily solvable clue. As a last resort, if there are multiple clues with the same low mine probability which have the same minimal number of unknown neighbors, we just take the first one which shows up in the list. Results and graphs on the next page.

As can be seen in the result graphs below, the advanced selection mechanism gives us an insane boost to average score while also decreasing the average solving time!

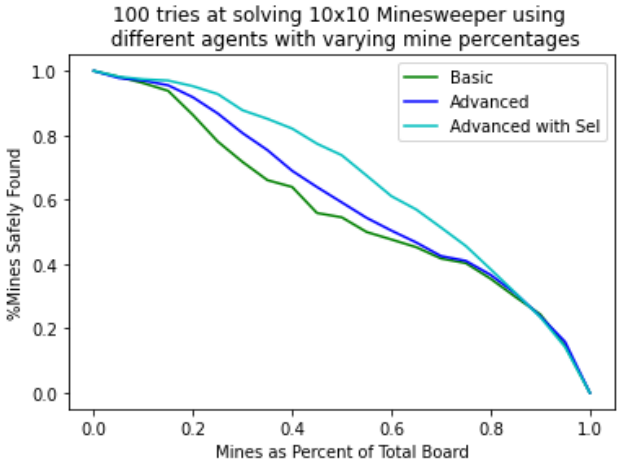

Figure 36: Agent Score Comparisons

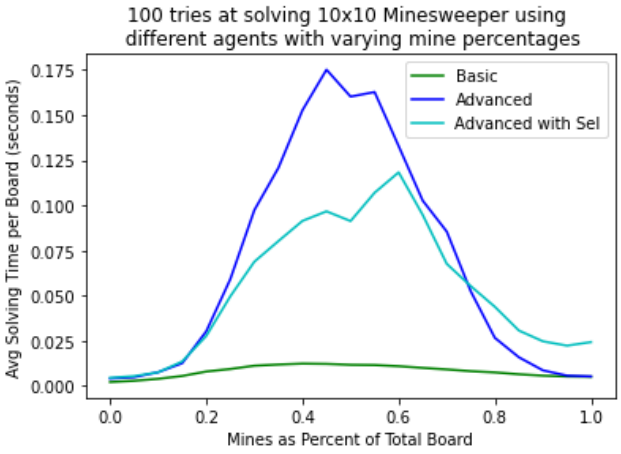

Figure 37: Agent Avg Time to Solve Comparisons

In addition, it's worth noting that as the dimensions of the board increases, the performance of the advanced agent with selection changes. As shown in the figures below, all versions of the advanced agents performance start to decrease the the board size increases. When the board is bigger, the number of choices for random selection increases, which makes it harder to perform inferences if the cells are chosen far from each other, forcing more random choices to be made overall. This also requires it to take more time as the algorithm is still performing the checks but not discovering any new information. However, the advanced agent with improved selection process still manages to triumph the advanced and basic agents in performance. The improved advanced agent's average solving time fluctuates between the advanced agent and the basic agent, meaning it runs faster than the normal random selection. In addition, since the "random selection" process is more specific, the total number of random moves decreased as the program is able to perform and discover more inferences than usual. This is especially helpful for larger board sizes as it enables the next random cell to be chosen to be in closer proximity to the knowledge we already know, which would allow new clues to be revealed at each step.

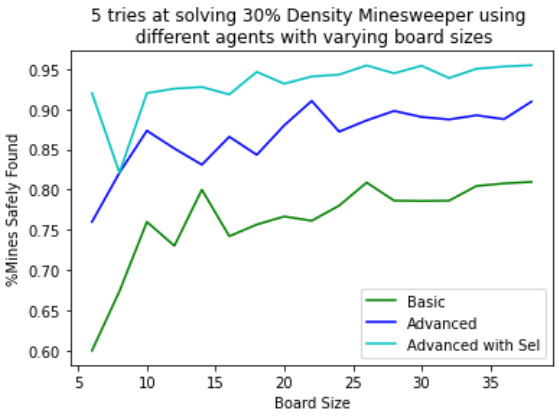

Figure 38: Agent Score Comparisons with 30% mine density over increasing board sizes

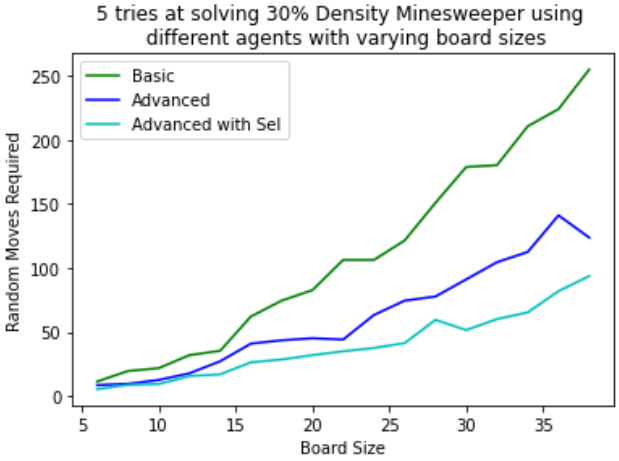

Figure 39: Random Moves Needed with 30% mine density over increasing board sizes

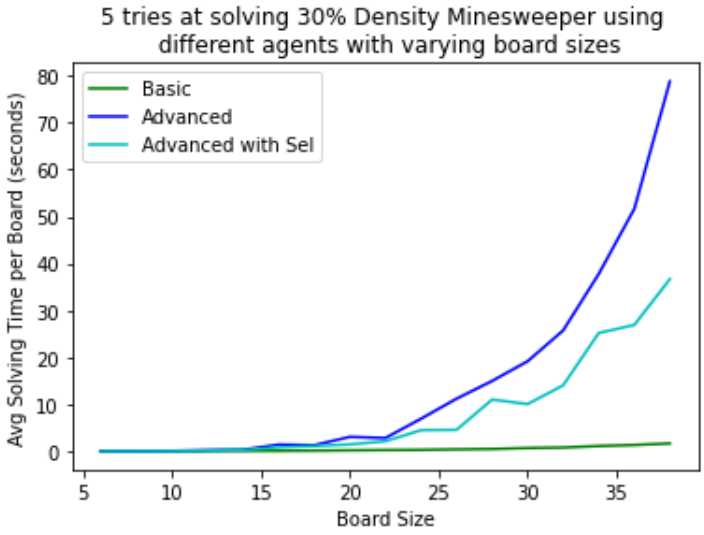

Figure 40: Agent Avg Time to Solve with 30% mine density over increasing board sizes

To finish out the bonus section, we also ran our code with the global information boost and the advanced selector on at the same time. You can see that the tradeoffs made by global information in terms of calculation time really boost the efficiency of the overall agent at mine densities higher than 75 percent. Those results are below.

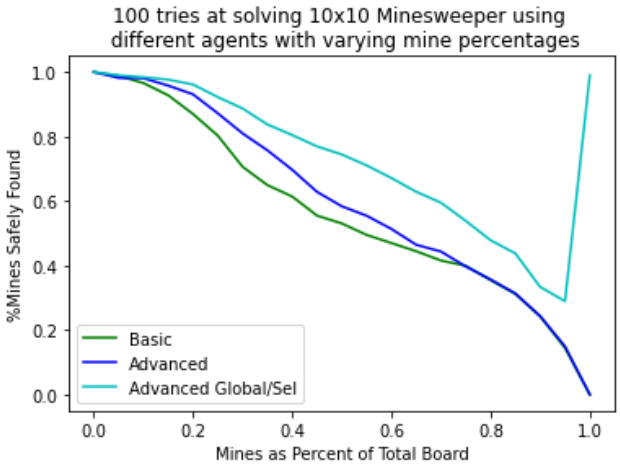

Figure 41: Agent Score Comparisons

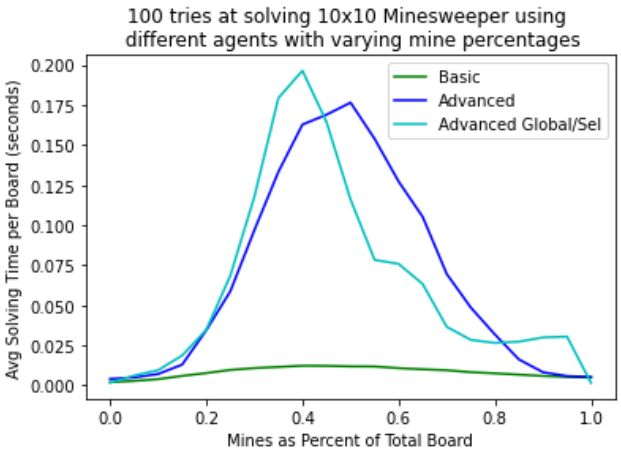

Figure 42: Agent Avg Time to Solve Comparisons