605.662 Data Visualization

Final Project - Final Paper

Sloan Cinkle
scinkle1@jh.edu

2 May 2022

Abstract

Wordle and Mastermind are both popular games in which a code-breaker is given a certain number of attempts to guess the correct combination of elements. Although the games operate over different domains, the shared concept of pattern-guessing between the two allows for similar optimization algorithms to be performed on both. Four optimization algorithms are explored on both games in this research, as well as a random guessing algorithm to provide baseline.

The final product of this research is two separate R Shiny applications which enable the user to run any number of simulations for either the Wordle or Mastermind game. The applications provide summary statistics for score, reduced solution space, and runtime of each algorithm, and displays histograms to visualize the data. Simulations which were run using these applications show evidence that, in some cases, it is beneficial for the code-breaker to select a guess that contradicts clues from previous guesses in order to eliminate the greatest number of potential solutions from the puzzle.

1 Introduction

Wordle is an online game invented by Brooklyn-based engineer Josh Wardle[1] which gained popularity in early 2022 as was soon acquired by The New York Times. The objective of the game is to guess the daily five-letter word within six guesses wherein the player receives feedback for each letter: if the letter is in the correct position, or in the incorrect position, or not in the word at all. Since the game's recent rise in popularity, it has attracted multitudes of programmers to create an optimal guessing algorithm to minimize the expected number of turns for the puzzle.

Another game from history which has a similar reputation for optimization is Mastermind[2], a code-breaking board game for two players. In this game, the code-maker begins by creating a combination of four colors from a pool of six available colors: red, blue, yellow, green, white, and black. The code can contain repeated colors if the code-maker chooses so. The code-breaker then has ten turns to guess the correct combination of colors, and after each guess, the code-maker reveals how many of the colors in the guess are in the correct position and how many of them are in the incorrect position. The difficulty in Mastermind compared to Wordle is that the code-breaker does not know which hints from the code-maker correspond to which elements of their previous guesses.

2 Background

WordleBot is a popular online application invented by Katz and Conlen[3], sponsored by The New York Times, which connects with Wordle to show the player a detailed breakdown of their daily game and to help improve player strategy. The application includes its own optimization algorithm to compare alongside the player's guesses, which uses information theory to choose the next word that will allow it to solve the game in as few steps as possible, assuming any of the remaining solutions are equally likely. The introduction of WordleBot sparked controversy and confusion for many players of the game, since oftentimes the algorithm appears to make a guess which contradicts previous hints in order to reduce the greatest number of potential solutions from the puzzle.

With the vast amount of public research already done for optimal strategies of the Mastermind game, I was able to find an academic paper which outlines some of the best strategies for code-breaking games which can be easily applied to a game of Wordle.

2.1 Algorithms

The algorithms that will be explored are Knuth's Worst Case, Kooi's Most Parts, Irving's Expected, and Neuwirth's Entropy, all as outlined by Miranda et al.[4] All of these algorithms optimize the code-breaker's guess on each turn beginning by constructing a partition table of potential guesses and potential feedback. Each column of the partition table represents a potential guess, and each row represents one of the possible combinations of clues received from the code-maker. The number of potential solutions for which a certain guess would give a certain combination of clues is counted in each element of the partition table.

[1] Wardle, Josh. "Wordle - a Daily Word Game." Wordle. The New York Times, April 21, 2022. Reference 6.

[2] Epp, Daniel. "Mastermind Game Rules," July 2014. Reference 2.

[3] Katz, Josh, and Matthew Conlen. "Introducing WordleBot, the Upshot's Daily Wordle Companion." The New York Times. The New York Times, April 7, 2022.

[4] Miranda, Luis O., Joel M. Quiles, and S. Kami Makki. "Optimized Strategies for the Mastermind Game." 4th Annual International Conferences on Computer Games, Multimedia and Allied Technology (CGAT 2011), 2011. Reference 4.

Answer	*AAAA*	*AAAB*	*AABB*	*AABC*	*ABCD*
0,0	625	256	256	81	16
0,1	0	308	256	276	152
0,2	0	61	96	222	312
0,3	0	0	16	44	136
0,4	0	0	1	2	9
1,0	500	317	256	182	108
1,1	0	156	208	230	252
1,2	0	27	36	84	132
1,3	0	0	0	4	8
2,0	150	123	114	105	96
2,1	0	24	32	40	48
2,2	0	3	4	5	6
3,0	20	20	20	20	20
4,0	1	1	1	1	1

Figure 1: Simplified partition table for the first turn of a **Mastermind** game.

Figure 1 shows a few columns from the partition table for the first turn of a Mastermind game where each letter (A, B, C, and D) represents a particular color. The potential hints that the code-breaker may receive from the code-maker are counted along the rows of the partition table for every potential guess. A partition table for the Wordle game has a similar structure, except each column represents a word in the guessing set and the rows are permutations of possible hints (correct, in word, and not in word) for each of the five letters in the guess.

Knuth's Worst Case algorithm operates by selecting the guess with the lowest maximum in its column of the partition table. For the example in Figure 1, this algorithm would choose AABB because that guess is guaranteed to result in less potential solutions for the puzzle on the next turn compared to the worst-case scenario of any other guess.

Kooi's Most Parts algorithm optimizes each guess by selecting the guess that has the most possibilities for potential hints. On each turn, the code-breaker selects the guess with the least number of zeros in its column of the partition table. For the example in figure 1, this algorithm would choose AABC or ABCD because neither of those columns contain any zeros, in other words, the code-breaker has a chance of receiving any possible set of clues from the code-maker with those guesses.

Irving's Expected algorithm calculates the expected solution size of each potential guess and selects the guess which minimizes that statistic. The equation for expected solution size is shown in Figure 2.

$$E(x) = \sum_{a_i} x \bullet P(a_i) = \sum \frac{x(a_i)^2}{1296}$$

Where,
$E(x)$ = expected size
x = type of guess the code breaker could ask
a_i = possible answers the code breaker can receive from the code maker. After analyzing the data from Table 1, for the first question the expected sizes are:

	AAAA	AAAB	AABB	AABC	ABCD
Expected size of a partition element	511.9	235.9	204.5	185.3	188.2

Figure 2: Formula for calculating expected solution size of each guess.

In this equation, it needs to be noted that the number 1,296 does not represent a constant, but the sum of each column. For the first guess of the Mastermind game, this number is equal to 1,296 because that is the number of potential solutions when no feedback has been received from the code-maker. For the first turn of the Mastermind game, the Expected algorithm guesses AABC because it minimizes the expected solution size out of all potential guesses.

Last of the optimization algorithms, Neuwirth's Entropy maximizes entropy, or the information content, of each guess by performing the calculation in Figure 3 for each column of the partition table.

$$-\sum_{i=1}^{n} p_i \log(p_i)$$

Where p_i is the probability that a solution is within the partition i of n possible codes. After analyzing the data from Table I, for the first question the entropies are:

	AAAA	AAAB	AABB	AABC	ABCD
Entropy	1.498	2.693	2.885	3.044	3.057

Figure 3: Formula for calculating the entropy of each guess.

The Entropy algorithm guesses ABCD for the first turn of a Mastermind game because that guess generates the maximum entropy out of all potential guesses.

3 Approach

I plan to build animated dashboard of visualizations in R showing simulations of code-breaking algorithms on different games to measure their success and to explore the idea of an optimal target score. The algorithms will run on solution sets for the game Wordle and the game Mastermind, and will receive feedback as per the rule set of the respective game.

I also wish to include simple random guessing as one of the algorithms in my application in order to provide a baseline for the more sophisticated algorithms. The random guessing algorithm simply returns a random guess from the set of potential guesses, which is equal to the set of potential solutions for targeted turns or the entire list of guessable words or combinations for untargeted turns.

3.1 Simulator

Sample code for a working model of Wordle in R Shiny is hosted by Chang and Strayer[5] on Github, and a video tutorial for the stylization of the application is demonstrated by Strayer[6] and hosted by RStudio on YouTube. This application and code provided a fantastic basis for the visual style of the application; however, it operates exactly like the original Wordle game in which the user inputs letters using a keyboard and submits a guess on each turn to gather information and ultimately solve the puzzle.

For the purposes of this project, it was necessary to connect this application to a computational agent which has access to previous guesses and feedback from the code-maker. This agent will analyze all of the information available at a certain point in the puzzle and return the optimal guess based on a given optimization algorithm. The simulator should take the optimal guess, display it with additional feedback, then prompt the agent for another guess, until the puzzle is complete or the maximum number of guesses have been made.

I added an action button to the application labeled Run simulations which toggles the user's view between the keyboard and option menu to run simulations. This prevents the user from inputting any of their own guesses, so only the agent can make guesses when simulation options are displayed. Options for the simulator include the type of algorithm to be ran, the number of iterations to be ran, and the target score for the simulator (described below). After enough guesses have been made by the agent, histograms and summary statistics for the simulations are displayed below the options menu.

[5]Chang, Winston, and Nick Strayer. "Shiny Wordle," February 8, 2022. Reference 1.
[6]Strayer, Nick. *Part IV: Styling a Shiny Wordle App with CSS. YouTube.* RStudio, 2022. Reference 5.

The user has the option to use one of the available algorithms to make a single guess on a puzzle. The Make a Guess action button makes a guess the same way that a real iteration of the simulation would, except the runtime statistic is not measured or recorded when the game terminates. When the user is not running simulations, the game operates as normal with an extra Show details action button at the bottom of the page. If this is chosen, a text box will be displayed which shows how the player's last guess divided the solution space of the puzzle: the number of potential solutions remaining and the percentage of potential solutions eliminated by the last guess.

After the Wordle simulator was complete, I made a complete copy of the application and implemented all the necessary changes to transform the game of Wordle into the game of Mastermind. First, rather than displaying each letter of the guess inside of a colored box, the letter is hidden and the box is colored to represent the letter it holds. The keyboard was reduced to only include six letters represented by the colors red, blue, yellow, green, white, and black. Instead of coloring the box to show which letters are correct, I placed a text box next to each of the previous guesses to show the number of colors in the correct position on top and the number of colors in the incorrect position on bottom. The list of words from which the solution is drawn was changed to a list of permutations of length 4 from a pool of 6 components. The length of a guess was changed to 4 and the maximum number of turns for the user was changed to 10. All of the action buttons, simulator options, and statistical and graphical output are mostly the same between the two applications.

3.2 Feedback

The Wordle game allows the player to make guesses from a total list of 12,972 words. Feedback from the code-maker is given after each turn for each of the letters in the code-breaker's guess: correct, in word, or not in word. The code-breaker continues to gather hints in this fashion until the correct word is guessed (win for the code-breaker) or until the code-breaker makes six incorrect guesses (win for the code-maker).

The Mastermind game begins by selecting a combination of four colors from a pool of six possible colors (red, blue, yellow, green, white, and black) with repetition if the code-maker chooses to do so. This results in a total of 1,296 potential solutions for the code-maker to choose from. The code-breaker makes a guess on each turn, and feedback is received from the code-maker as the number of colors in the correct position and the number of colors in the incorrect position. The code-breaker does not know which hint corresponds to which color. The code-breaker continues to make guesses and receive feedback until all four colors are correct (win for the code-breaker) or until the code-breaker makes ten incorrect guesses (win for the code-maker).

3.3 Statistics

Three statistics which will be measured and graphed concurrently as simulations take place are score, reduced solution space, and time. Histograms for each of these statistics will be constructed and displayed below the simulator, and the median, average, and standard deviation of each statistic will be calculated and displayed below each respective chart.

The score for each iteration of the games will be fairly simple to measure. The score of the code-breaker is the number of turns that it takes for the code-breaker to guess the correct word or combination. If the code is not guessed correctly before the turn limit, the code breaker is given a score of the maximum number of turns plus one.

The second statistics to be measured is the reduced solution space of the puzzle after each guess made by the code-breaker. This is the percentage of potential solutions that were eliminated from the puzzle by the last guess. This statistic is different from the others because it is recorded after every turn rather than at the end of each game. For this reason, the space histogram and statistics are displayed above the others to reduce empty space on the dashboard if the first iteration of the simulator is not yet complete.

Lastly, I would like to measure the amount of time that it takes for the algorithms to solve each puzzle, since I predict that the untargeted guesses will take much longer to optimize than targeted guesses. I also

predict that more successful algorithms in terms of score and space will have a lower median and average runtime than other algorithms, since they filter more solutions out of the potential solution set after each guess and solve the puzzle in less turns.

3.4 Target Score

An additional option which will be included in the simulator is to implement a target score for the algorithm. If the target score is set to three, then the algorithm will be forced to make a targeted guess on the third turn and for all future turns. For turns in which the algorithm makes a targeted guess, the set of potential guesses is filtered to be equal to the list of potential solutions for all previous feedback received from the code-maker. The result is that, on targeted turns, the algorithm is unable to make a guess that does not have a chance of solving the puzzle. In Wordle, this is typically known as playing on Hard Mode.

The benefit of making an untargeted guess is that it may be possible to eliminate more potential solutions of the puzzle by making a guess that does not actually have a chance of solving the puzzle. The obvious downside to this strategy is that any guess which does not fall into the set of potential solutions has no chance of solving the puzzle, and another is that the algorithms are bound to take much longer optimizing untargeted guesses because they must consider the entire list of words or combinations on every turn, resulting in a much larger partition table at each call of the algorithm.

My prediction is that all algorithms will perform better in terms of score and space when the target score is set above the minimum value but below the average score for the respective algorithm when the target is set to the minimum value. My basis for this hypothesis is that an untargeted second guess should have a higher chance of eliminating more potential solutions than a targeted second guess. The limitation of this strategy is that it typically reduces the chance of completing the puzzle on the second guess.

The target score is represented as a slider input in the Shiny application. It has a minimum value of two; setting the target to one would have no effect on the set of potential guesses since there is no feedback from the code-maker at the first guess. The maximum value of the slider is the maximum number of turns plus one, at which point the code-breaker will not filter the set of potential guesses to potential solutions for any of its guesses.

3.5 Shortcutting

The actual Wordle game has a list of predefined solutions[7] going up to October 20, 2027 that can be found online. However, the list of all guessable words is much longer, since many five-letter words are valid words but too obscure to be a good daily word for the global puzzle. The total counts are 2,315 eventual solutions for the game and 12,972 total guesses that the player can choose from. In order to reduce the solution set and guessing set of each algorithm to a feasible amount, the code-maker and the guessing algorithms implemented in my application only consider words which are actual solutions to the Wordle puzzle. However, when the user is playing the game with the keyboard and not running simulations, they are able to choose from the entire list of guessable words.

Due to the fact that no feedback has yet been received when the code-breaker makes its first guess, it quickly became apparent that performing thousands of simulations for games would take an unrealistic amount of time simply from optimizing each algorithm's first guess. Another point is that, with no feedback yet received by the code-maker, each algorithm is always going to choose the same first guess. It is possible to save massive amounts of computation time by pre-defining the first guess made by each algorithm. After this was implemented, the expected runtime of my desired number of simulations was still unrealistic for untargeted guesses.

Creating the partition table needs to be performed at the start of each guess and has a runtime efficiency of O(potential solutions $*$ potential guesses). For an untargeted second guess, this is still a massive number

[7] Wardle, Josh. "Wordle Answers," January 9, 2022. Reference 7.

of computations. I furthered my solution to create a shortcut for the second guess of each algorithm as well as the first. I ran simulations for the second guess of each algorithm using the algorithm's best first guess and every possible set of feedback that could be received from the code-maker. The result is the best second guess of each algorithm depending on the feedback received by the first. This data is stored in wordle/untarget.Rdata and mastermind/untarget.Rdata. With the code already written to run these simulations, I did the same for a targeted second guess of each algorithm and stored the results in wordle/target.Rdata and mastermind/target.Rdata. These files are accessed by the application upon opening so that all algorithms can save time on the first and second guesses of repeated simulations. An algorithm will not use this data if the first guess of the puzzle is not equal to its preferred first guess. An impact of this on the output is that the average runtime of each iteration should increase drastically when the target score is increased above three, except for random guessing.

4 Results

The number of iterations ran for the Wordle and Mastermind games are shown in Figure 4. Each iteration represents one game from start to finish. Results for medians, averages, and standard deviations of scores, reduced solution spaces, and runtimes of each algorithm is shown in Figure 5 for the Wordle game and in Figure 6 for the Mastermind game. Histograms for all simulations are shown in Figures 7 through 13 with Random Guessing, Knuth's Worst Case, Kooi's Most Parts, Irving's Expected, and Neuwirth's Entropy from left to right.

For the Wordle game, Kooi's Most Parts algorithm performed definitively better than all other algorithms, in terms of score. For simulations where the target score is set to the minimum value, the Most Parts algorithm is the only one that results in a median score of three – all others have a median score of four. For simulations where the target score is set to three, the Most Parts, Expected, and Entropy algorithms all resulted in a median score of three. None of the algorithms performed as well when the target score is set to the maximum value.

As I predicted, setting the target score equal to three resulted in a lower average score than setting the target score to its minimum value for every optimization algorithm besides random guessing. To scientifically prove this would require more statistical testing and computation on the averages and standard deviations, but as is stands, the results of the application seem to be fairly conclusive by themselves.

When considering medians, the algorithms that were most successful in terms of score seem to be the same as the ones that were successful in reducing the solution space of the puzzle. However, the averages of these statistics do not show as conclusive results. Figures 7, 8, and 9 show that as the target score increases, the percentage of potential solutions eliminated from the puzzle drift towards 0% and 100%, whereas simulations with a low target score show more guesses which reduced a moderate amount of the solution space around 50%.

For the Mastermind game, the Most Parts, Expected, and Entropy all performed equally well for the median score over all iterations. These algorithms all resulted in a median score of four for simulations with a target score less than its maximum value, and a median score of five for completely untargeted simulations. The Most Parts algorithm stands out as receiving a slightly higher score than all other algorithms on average, but these results are not nearly as conclusive as those we had for the Wordle game. Two observations that do seem to be grounded from Figure 6 are that the Worst Case algorithm did not score as well as the other algorithms and that simulations with a maximum target score did not score as well as simulations with a lower target score.

Figures 7, 8, and 9 show that all algorithms on the Wordle game mostly reduce between 95% to 100% of the solution space with each guess. This is shown by a maximum on the reduced solution space histograms at the right-most bin on the x-axis. Figures 10, 11, 12, and 12 show that all algorithms on the Mastermind game mostly reduce around 80% of the solution space with each guess. This difference is most likely due to the structure of each game's guess set and solution set and the level of feedback provided by the code-maker

after each guess.

Similar to the Wordle game, Figures 10, 11, 12, and 12 show that, in the Mastermind game, running simulations with a higher target score results in higher maxima on the edges and lower minima in the center of the histograms for reduced solution space. This is due to the fact that the set of potential guesses is not filtered for valid solutions before optimization is performed, so there is a possibility that a word that does not fall into the set of potential solutions eliminates more potential solutions than one that does.

As expected, simulations with a target score above three show a huge increase in median and average computation time over all iterations of the algorithm, between 15 to 20 seconds on average. I found it funny that the iterations for the untargeted random guessing took longer on average than the targeted random guessing, because even though the untargeted algorithm doesn't require any filtering of potential guesses at all, the fact that the simulator lost nearly every game and had to display more incorrect guesses completely counteracted the time that it saved from not having to filter data.

Another interesting observation I made which is not supported by the Figures is that simulations of random guessing with a target score equal to the maximum number of turns usually resulted in a win for the code-breaker. This means that the code-breaker can disregard all previous clues and make completely random guesses for every turn except for the last one, and there will usually be only one potential solution left to solve the puzzle. The win-rate of this strategy is higher for the Mastermind game than for the Wordle game, most likely because the algorithm gets more chances to explore the puzzle with nine turns in Mastermind and five turns in Wordle.

5 Conclusion

Overall, I was extremely satisfied with the final product of my applications. As someone with no prior experience in either Javascript nor CSS, I was ecstatic with of my accomplishment of transforming the Wordle application into a working model of Mastermind, not to mention implementing an entire simulator to animate results from the algorithms in real-time for both. I was preparing myself to compromise on many aspects of my original vision which I had thought to be unrealistic, but as I spent more time improving all different aspects of the application, I oftentimes found solutions for small issues that I had already decided to move past.

I was somewhat disappointed that I was not able to use the entire list of Wordle's guessable words in my algorithms, since it would have taken an unrealistic amount of time to generate a partition table for the entire list of 12,972 guessable words. In the future, it would be awesome to invent a more efficient method for constructing the partition table so that the algorithms have many more options to explore.

I am satisfied with my results from these simulations for the time being, but in the future, I would love to scientifically explore the results of my simulations to statistically prove that setting a target score below the algorithm's average score typically results in the least number of attempts needed to guess the correct pattern. To further this statistical analysis, it would also be interesting to plot various distributions over each of the histograms in my output to determine which distributions fit each algorithm best.

Bibliography

1. Chang, Winston, and Nick Strayer. "Shiny Wordle," February 8, 2022. https://github.com/wch/shiny-wordle.

2. Epp, Daniel. "Mastermind Game Rules," July 2014. https://magisterrex.files.wordpress.com/2014/07/mastermindrules.pdf.

3. Katz, Josh, and Matthew Conlen. "Introducing WordleBot, the Upshot's Daily Wordle Companion." The New York Times. The New York Times, April 7, 2022. https://www.nytimes.com/2022/04/07/upshot/wordle-bot-introduction.html.

4. Miranda, Luis O., Joel M. Quiles, and S. Kami Makki. "Optimized Strategies for the Mastermind Game." 4th Annual International Conferences on Computer Games, Multimedia and Allied Technology (CGAT 2011), 2011. https://dl4.globalstf.org/products-page/proceedings/cgat/optimized-strategies-for-the-mastermind-game/.

5. Strayer, Nick. *Part IV: Styling a Shiny Wordle App with CSS. YouTube.* RStudio, 2022. https://www.youtube.com/watch?v=4_tX4IUDJZ0.

6. Wardle, Josh. "Wordle - a Daily Word Game." Wordle. The New York Times, April 21, 2022. https://www.nytimes.com/games/wordle/index.html.

7. Wardle, Josh. "Wordle Answers," January 9, 2022. https://paste.ee/p/4zigF.

Appendix

	Iterations				
Wordle	Random	Worst Case	Most Parts	Expected	Entropy
Target = 2	5000	2000	2000	2000	2000
Target = 3	2000	1000	1000	1000	1000
Target = 7	5000	1000	1000	1000	1000
Mastermind					
Target = 2	5000	2000	2000	2000	2000
Target = 3	2000	1000	1000	1000	1000
Target = 4	2000	1000	1000	1000	1000
Target = 7	5000	1000	1000	1000	1000

Figure 4: Iterations for all instances of simulations ran for the following output.

	Wordle Score					Wordle Reduced Solution Space					Wordle Time				
Median	Random	Worst Case	Most Parts	Expected	Entropy	Random	Worst Case	Most Parts	Expected	Entropy	Random	Worst Case	Most Parts	Expected	Entropy
Target = 2	4	4	**3**	4	4	80.6	85.7	**87.5**	85.7	85.7	0.9508	0.6193	**0.5378**	0.5751	0.5707
Target = 3	4	4	**3**	**3**	**3**	78	87.5	**89.4**	**88.5**	**88.4**	1.0649	1.5168	**1.3917**	**1.2148**	**1.4181**
Target = 7	7	4	4	4	4	50	85.7	88.09	87.5	87.9	1.5082	26.9172	25.2637	25.3123	25.5221
Average															
Target = 2	4.06	3.67	**3.53**	3.62	3.62	67.3	68	**67.6**	67.8	67.5	1.0205	0.6097	**0.5575**	0.5759	0.5793
Target = 3	4.33	3.66	**3.49**	**3.55**	**3.59**	65.7	67.8	**66.8**	**67.3**	**67.6**	1.1243	1.5439	**1.4748**	**1.3264**	**1.4358**
Target = 7	6.99	3.82	3.61	3.72	3.7	47.5	63.8	64	64	63.9	1.6014	22.8469	17.0896	19.4646	16.1417
Standard Deviation															
Target = 2	1.03	0.87	**0.82**	0.85	0.85	33.8	36	**37.3**	36.4	36.6	0.419	0.2237	**0.1969**	0.2005	0.199
Target = 3	0.95	0.78	**0.71**	**0.74**	**0.75**	33.4	36.7	**38.9**	**38**	**37.6**	0.3677	0.7736	**0.7217**	**0.4795**	**0.5459**
Target = 7	0.22	0.66	0.66	0.6	0.62	39.1	39.9	41.1	40.6	40.8	0.3395	16.5097	16.088	14.5668	15.477

Figure 5: Statistics for all five algorithms on the **Wordle** game.

	Mastermind Score					Mastermind Reduced Solution Space					Mastermind Time				
Median	Random	Worst Case	Most Parts	Expected	Entropy	Random	Worst Case	Most Parts	Expected	Entropy	Random	Worst Case	Most Parts	Expected	Entropy
Target = 2	5	5	**4**	**4**	**4**	80	81.2	**82.3**	**82.3**	**80.6**	1.356	1.1249	**1.0383**	**1.0592**	**1.0609**
Target = 3	5	5	**4**	**4**	**4**	79.5	81.8	**82.3**	**82.3**	**80.6**	1.2756	3.9015	**3.5269**	**3.2293**	**3.4422**
Target = 4	5	5	**4**	**4**	**4**	77.4	81.2	**82.4**	**82.9**	**80.6**	1.384	19.7599	**19.6484**	**16.8524**	**19.5053**
Target = 7	11	5	5	5	5	0	81	82.3	82.4	80.6	2.7652	21.4234	17.3475	17.3593	17.6791
Average															
Target = 2	4.65	4.5	**4.38**	**4.4**	**4.44**	67.9	68.7	**68.9**	**69.2**	**68.7**	1.4296	1.1077	**1.0451**	**1.0503**	**1.0711**
Target = 3	4.71	4.47	**4.38**	**4.38**	**4.43**	67.8	69.5	**69**	**69.3**	**68.8**	1.32255	3.6508	**3.5065**	**3.2255**	**3.447**
Target = 4	5.01	4.54	**4.4**	**4.41**	**4.46**	65.7	68.5	**68**	**68.1**	**68**	1.4055	19.2849	**19.2076**	**16.5762**	**19.5053**
Target = 7	10.96	4.75	4.59	4.64	4.65	34.2	65	65.4	65.1	65	2.8538	18.8528	14.8062	14.9875	15.4592
Standard Deviation															
Target = 2	1.356	1.1249	**1.0383**	**1.0592**	**1.0609**	29.7	30.9	**31.8**	**31.1**	**31.5**	0.5105	0.326	**0.2799**	**0.2788**	**0.2897**
Target = 3	0.85	0.71	**0.74**	**0.75**	**0.7**	29.4	30.5	**31.7**	**31**	**31.4**	0.3676	1.8795	**1.6249**	**0.9709**	**1.235**
Target = 4	0.77	0.64	**0.61**	**0.61**	**0.64**	29.6	31.2	**33.2**	**32.9**	**32.4**	0.3666	3.0859	**3.1631**	**2.4495**	**2.5763**
Target = 7	0.52	0.53	0.59	0.56	0.58	37.9	34.6	35.3	35.2	35.1	0.6121	5.0625	4.8404	4.614	4.8093

Figure 6: Statistics for all five algorithms on the **Mastermind** game.

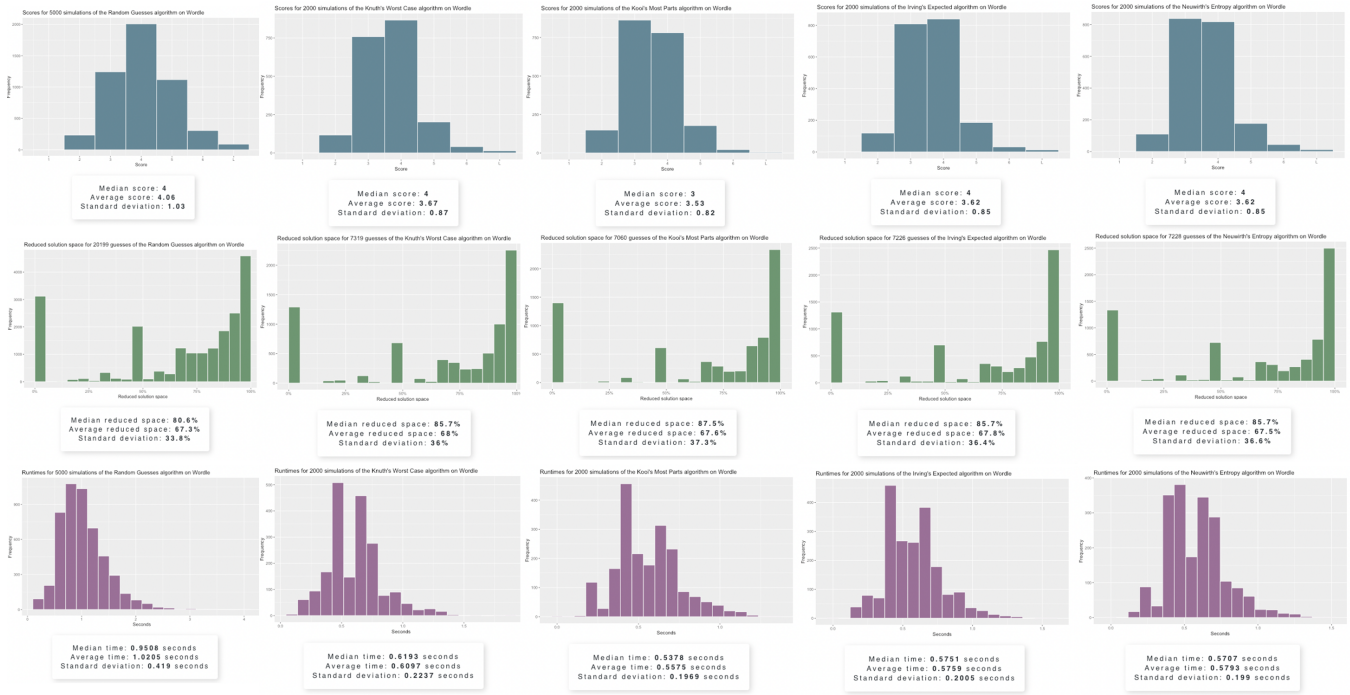

Figure 7: Score, space, and time output for all five algorithms on the **Wordle** game eliminating impossible solutions from the guess set after **every** guess.

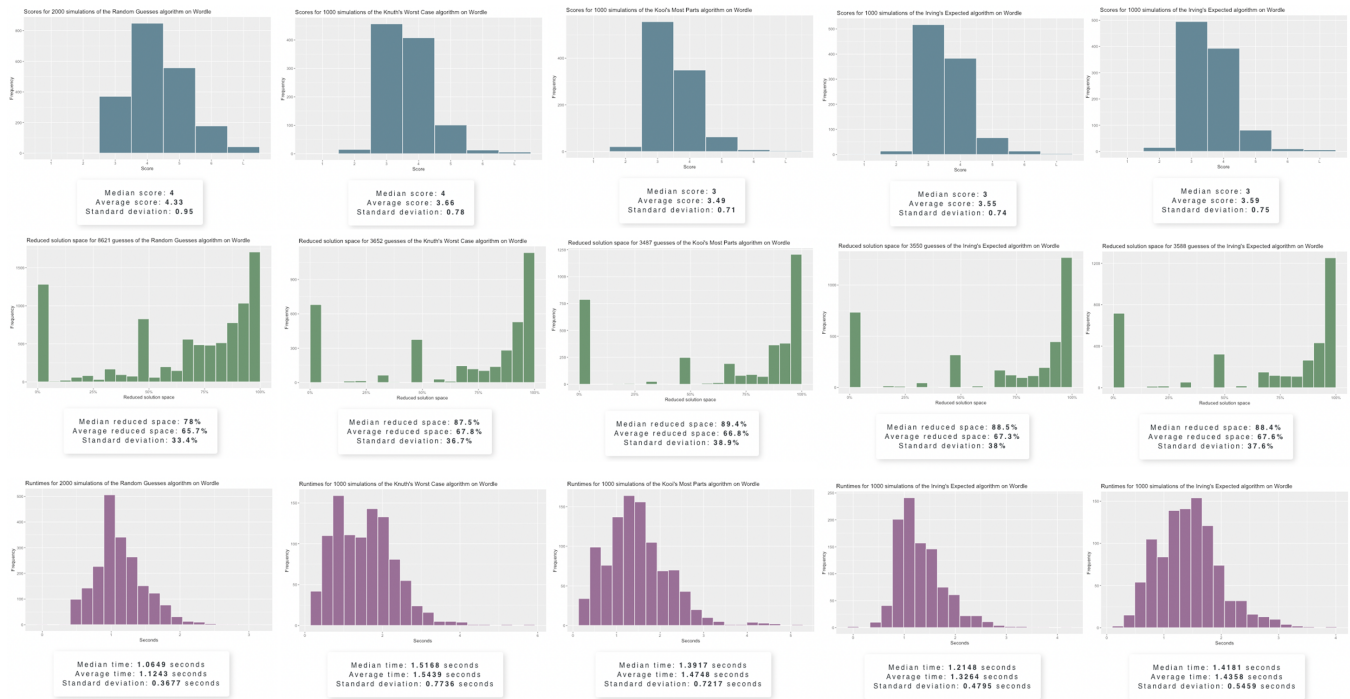

Figure 8: Score, space, and time output for all five algorithms on the **Wordle** game eliminating impossible solutions from the guess set after the **second** guess.

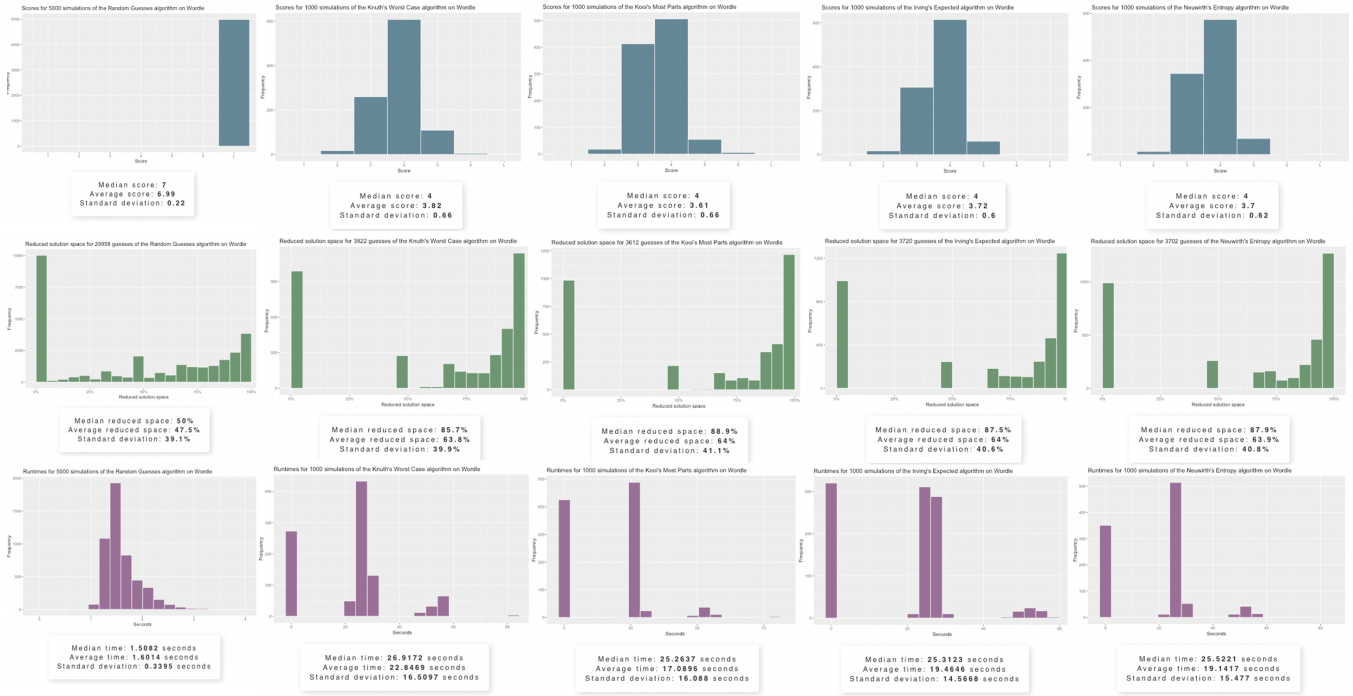

Figure 9: Score, space, and time output for all five algorithms on the **Wordle** game **without eliminating any** impossible solutions from the guess set.

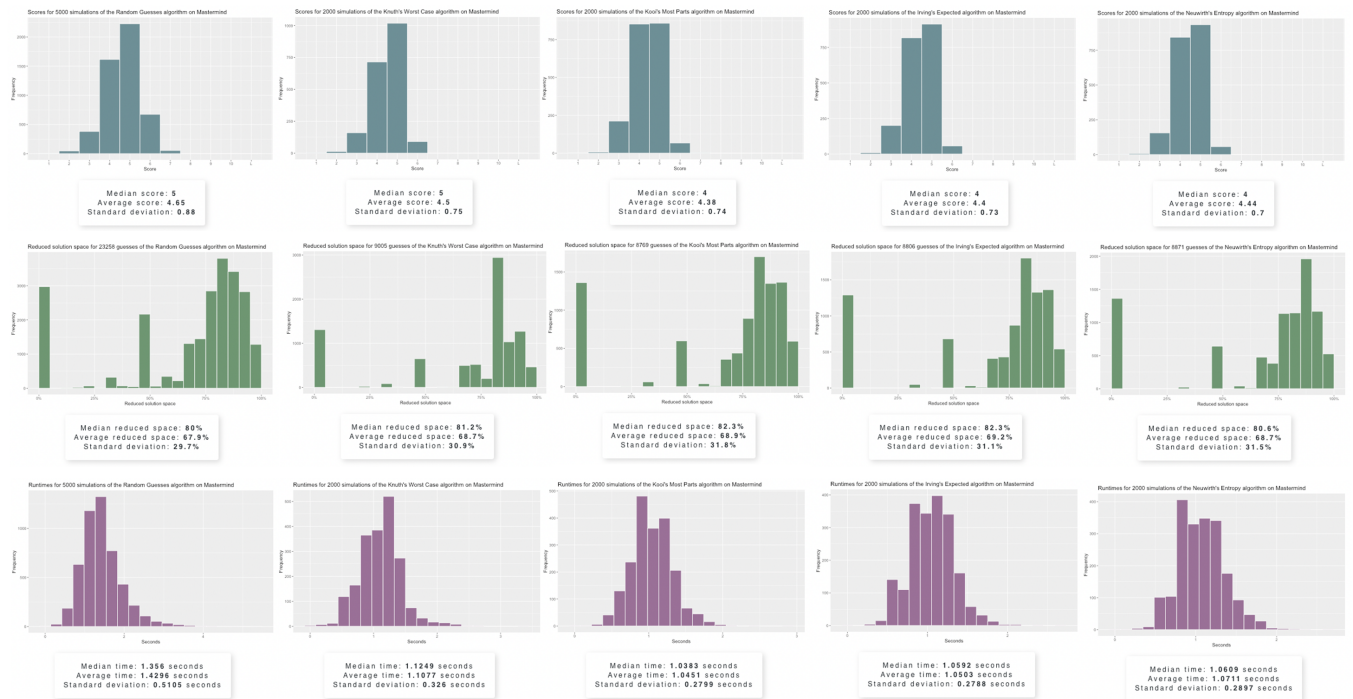

Figure 10: Score, space, and time output for all five algorithms on the **Mastermind** game eliminating impossible solutions from the guess set after **every** guess.

Figure 11: Score, space, and time output for all five algorithms on the **Mastermind** game eliminating impossible solutions from the guess set after the **second** guess.

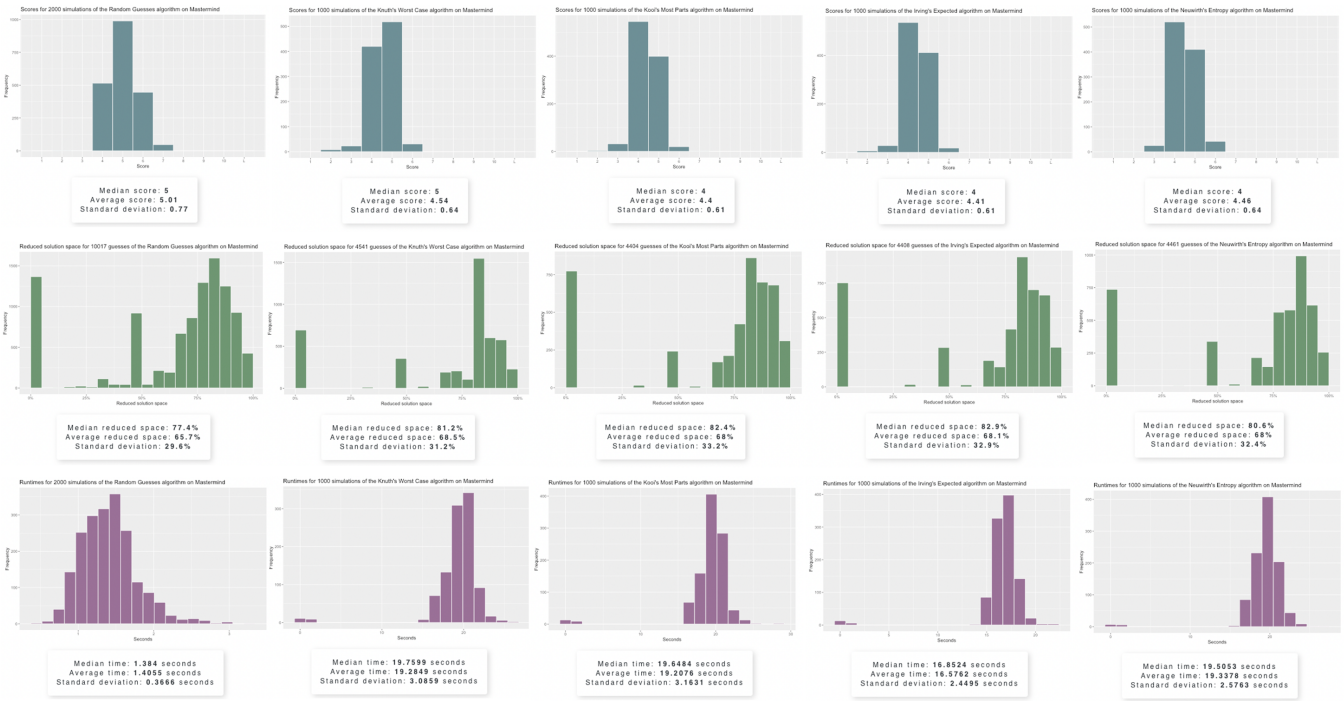

Figure 12: Score, space, and time output for all five algorithms on the **Mastermind** game eliminating impossible solutions from the guess set after the **third** guess

.

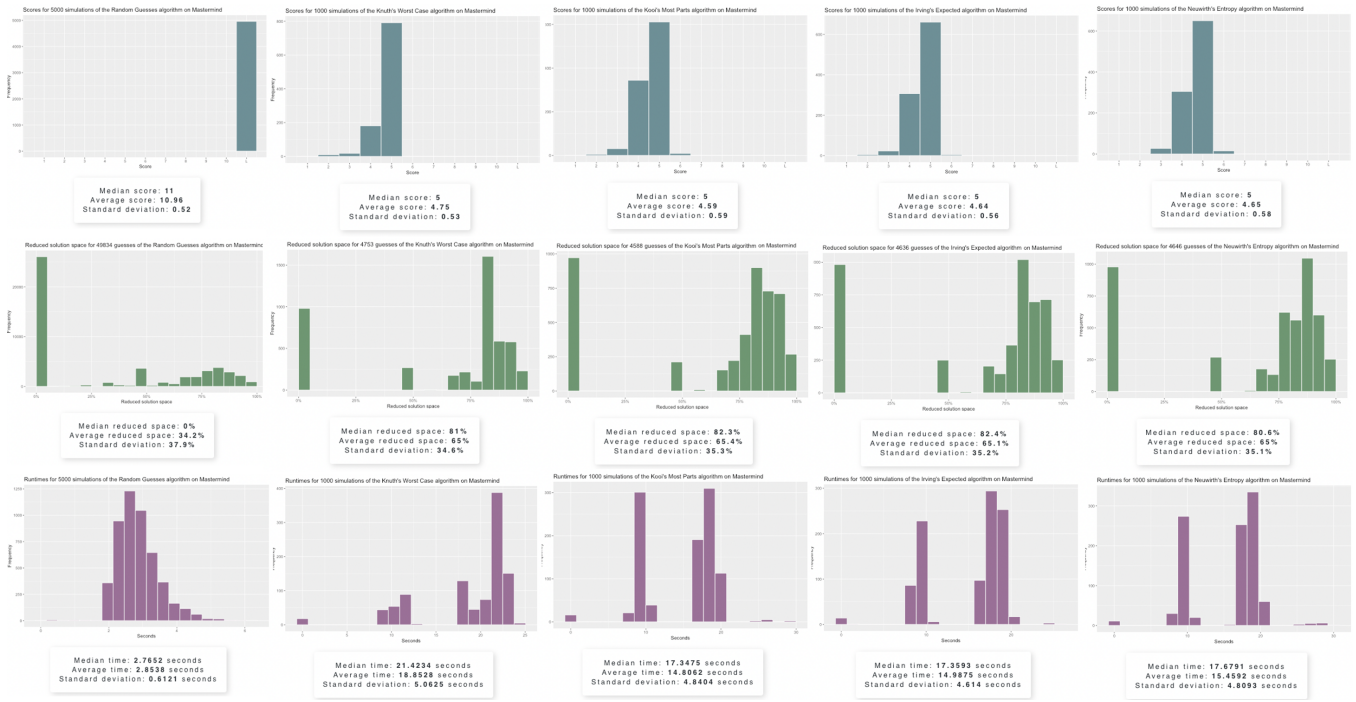

Figure 13: Score, space, and time output for all five algorithms on the **Mastermind** game **without eliminating any** impossible solutions from the guess set.