Analysis of Algorithmic Methods to Effectively Solve Wordle

Safwan Diwan and Mikkel Folting

December 14, 2022

1 Abstract

Wordle is a popular word game where players are given six attempts to guess a word. The player is given information regarding the solution after each guess that can be used to determine the solution word. In this study, nine algorithms that use different approaches to guess the solution word were created and analyzed. A comprehensive algorithm ranking system, based on three metrics (game win percentage, average guess count, and runtime) was built in order to evaluate the algorithms and compare them against each other. This process resulted in the conclusion that using two preset words for the first two guesses allows for not only the possible words dictionary to be effectively pruned but also key information necessary to determine the solution word to be gained. It was also noted that the combination of this strategy with a positional ranking system, where possible guess words are ranked based on a letter-by-letter comparison to the information gained from each guess, garnered the best results. This algorithm demonstrated a 93.8% win rate and an average of 4.560 guesses per game, a major improvement in comparison to the random guess algorithm that was used as a performance baseline and had a 71.0% win rate and an average guess count of 5.853.

2 Introduction

Wordle is a web-based game developed by Josh Wardle that, after gaining widespread popularity, was acquired in January 2022 by the New York Times (NYT). Wordle is quite unique in that it combines key components of Hangman and Mastermind to create a game that asks the player to guess a five-letter word in six attempts. After each guess, the user is presented with a result that highlights the letters in the word that are in the correct position in comparison to the solution, as well as the letters that are in the word but in the incorrect position. This allows the player to use information from each guess in order to determine the solution word. The original version of the game has 2,315 possible solution words and 10,657 additional words that are accepted as guesses.[2] A screenshot displaying a sample playthrough for the game is shown below in Figure 1. While the game vocabulary under the NYT has shifted slightly due to the political ramifications of using some words, the majority remains the same. It should also be noted that the game also features a "hard mode" setting. When enabled, this requires the player to only guess words that utilize the

information gained from previous guesses. For example, if the word BEARS was guessed and it was noted that the letter S was in the word but in the incorrect position, then the subsequent guess would have to include the letter S in any position except the last one.

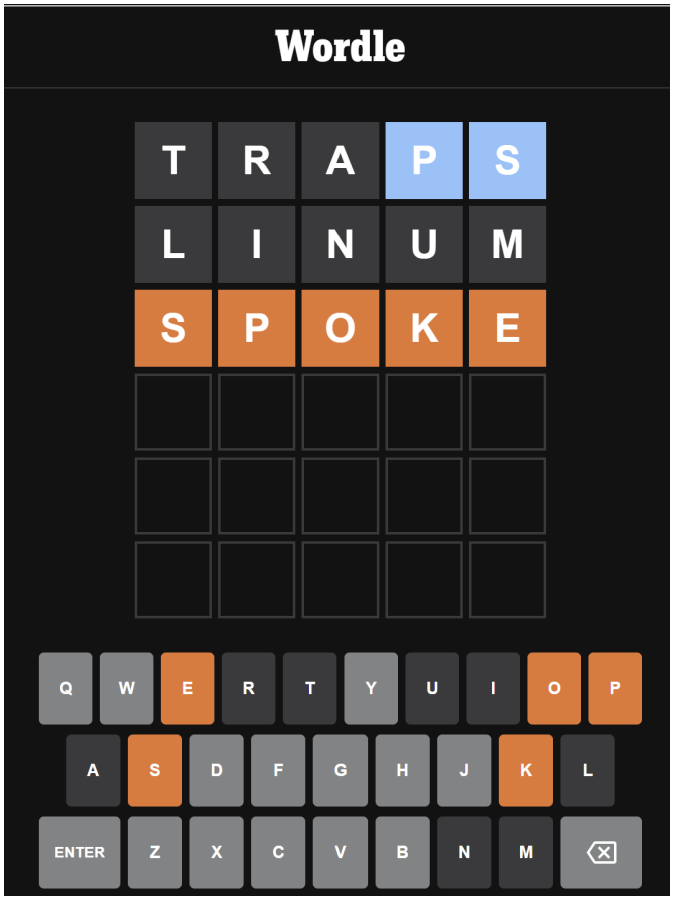

Figure 1: An example playthrough of the Wordle web game on the New York Times website. Orange letters indicate letters in the correct spot, blue letters indicate letters in the word but in the incorrect spot, and black letters demonstrate letters not present in the solution word.

There have been a number of studies that discuss using artificial intelligence techniques to efficiently solve Wordle, but, as the game has only recently gained popularity, many of these studies have not been peer-reviewed. The game Mastermind, however, is quite similar to Wordle and has been studied extensively. In both games, the player has a set number of guesses to guess the answer to a riddle. The key differences revolve around the fact that instead of letters that make words, the correct answer contains four colors in a set order. The player also does not know which color is correct/in the right position, they just know the number of these occurrences in their guess. As the game has a clear correct answer and increased restraints on possible guesses as more color combinations are guessed, this further

exemplifies the fact that the game is quite similar to Wordle, and therefore approaches to efficiently solve both games can be noted as similar.

The same is true for the classic AI constraint-satisfaction problem (CSP) of cryptarithmetic. This can be used as a template for solving Wordle as this game can be thought of as a CSP. The constraints present in Wordle consist of the letters and their possible positions, as each guess provides information on not only the identity of the correct letters but also the location of these letters. Therefore, after each guess, the domain space of each of the letters is updated according to the newly learned constraints. As cryptarithmetic uses a form of constraint-based searching that can be applied to a number of different problems, strategies used from this puzzle can be used to determine effective methods to solve Wordle.

Overall, this game poses an interesting challenge in regards to finding a method to determine the solution in as few guesses as possible. This is because there are a number of possible algorithms that can be used to determine the next guess, all of which will likely have differing performances. The strategy for each guess can use a different strategy, and the overall similarity of this game to a number of other puzzles allows for possible solution methods to be easily generated but used in novel ways. In addition, the widespread popularity of the game makes it an intriguing choice as it allows for an application of artificial intelligence techniques learned in the classroom on a real-world problem/game.

3 Related Work

3.1 Mastermind

One of the most common types of algorithms that have been used in a number of studies to find efficient solutions for Mastermind are evolutionary and genetic algorithms. In one study, Merelo-Guervós et al. discuss using a stepwise optimal algorithm that attempts to find and play a guess that meets all the hints.[11] Each possible guess is given a score based on its consistency with the correct answer and this score is used as the fitness. To ensure diversity is changed every generation, three different changes are completed on each combination. The first change occurs 20% of the time and consists of simply a permutation on the combination. The second and third changes each occur 40% of the time and consist of a mutation that replaces one of the colors with the next one in the combination and a crossover that interchanges the middle segments of two combinations. It should also be noted that in each generation 50% of the combinations (the top half in terms of fitness) are retained. The results of running this algorithm were noted to outperform previously defined evolutionary algorithms in terms of the number of guesses required to reach the solution.

More recently, Leus et al. discuss an algorithm that begins by using the same fixed guess used by Merelo-Guervós and was determined to reduce the average number of guesses by 0.02.[10] The algorithm creates new generations through either a 1-point or 2-point crossover from two parent combinations. These crossovers each have a 50% probability of occurring. There is also a 3% probability that a mutation will follow the crossover, replacing a random position with a random color. In addition, there is a 3% chance that two colors at random positions in the combination can be swapped. Finally, there is a 2% chance that two positions

in the combination are randomly chosen and the order of the colors between these positions is inverted. It was noted that this solution is comparable to other algorithms but yielded worse results than detailed in the study by Merelo-Guervós et al. However, the differences in the statistical analyses between the two studies may not have allowed for an accurate comparison.

Another method used is the k-way-branching algorithm (KWB). This algorithm is simply a beam search that only considers k guesses at each node. Chen et al. discuss this in a study that uses a hash function that is based on a heuristic to cluster the possible guesses.[7] For example, in the worst-case scenario, the heuristic used aims to minimize the number of candidates in the larger response classes after each guess. A response class is simply a set of guesses that result in the same consequences. The heuristic, in this case, will attempt to minimize these response classes, thereby aiming to minimize the height of the game tree. The expected case, however, uses a different heuristic. In this case, the goal is to maximize the entropy by clustering the remaining candidates after a guess. This allows the hash function to minimize the external path length of the game tree.

A two phase optimization algorithm (TPOA) can also be used as detailed in another study authored by Chen et al.[6] This technique uses the same KWB algorithm as mentioned previously and therefore combines multi-way search, sampling, and clustering techniques. The TPOA tree is split up into two phases, exploration and exploitation. Exploration attempts to discover useful partial solutions and occurs until a certain depth is reached. After this depth, exploitation occurs using a greedy search to find the best complete solution. In this scenario, annealing can be used to decrease the branching factor. In addition, if a set number of branches is considered during the exploration phase and the other branches are ignored, the TPOA tree can be pruned, increasing efficiency. This is necessary for large search spaces such as that defined by Mastermind. After a set of guesses is determined, the KWB algorithm can be used to evaluate the next best guess. A* search was also compared, however, it was noted that TPOA using annealing was far more efficient in terms of runtime and resulted in the same average guesses to find a solution. In addition, the results demonstrated that the use of the KWB algorithm consistently obtained better results than those mentioned in the latest evolutionary algorithm study at the time of writing.

3.2 Cryptarithmetic

Cryptarithmetic is a logic puzzle involving an arithmetic expression that has been converted to letters. There are constraints in the game - each letter represents one digit between 0-9, no two letters can represent the same digit, and there are no leading zeros.[16] From this, an AI agent that uses CSP can be built to solve this game. Some approaches to this are discussed below.

Similar to solutions to Mastermind, the common way to approach the cryptarithmetic problem is by building a parallel genetic algorithm (PGA) for the CSP representing the problem. Sharma et al. discuss one such approach in which a PGA is constructed that builds chromosomes of candidate solutions and evolves those towards the desired solution. Random mutations are implemented that help prevent the search from getting trapped in

regions where a locally optimal state occurs and represent the concepts of chromosomes, genes, and nucleotides with symbols (functions, variables, or constants). In this case, a chromosome is represented as a series of possible values that each letter in each position could have. Using these ideas, the problem can be solved by creating chromosomes and then evolving them towards better possible chromosomes by randomly mutating and trying different approaches. No exact numbers are given, but the paper mentions that the PGA approach consistently outperforms other traditional approaches of depth or breadth-first search. [16]

A similar approach is discussed in a study by Rahnavard and Dastghaibyfard. This study concurs with the results by Sharma et al. as it is noted that PGA performs very well against large-scale CSPs such as cryptarithmetic. A similar chromosome representation is used, assigning each possible digit (0-9) to a letter being used in the puzzle, and if there are more digits than there are letters being used, then assigning the rest of the digits to empty letters. Mutations involve possible swaps of digit assignments, while the fitness function simply calculates the sum of the current assignment of digits to letters and sees how close it is to the assignment of digits to the letters in the answer to the arithmetic expression. Mutations then create the next generation of chromosomes and only the "fittest" of these will be used for the next set of mutations. To parallelize, the new populations are spread among multiple threads and re-distributed after finding the next fittest population. Upon testing, the authors found that the PGA approach was significantly faster than the traditional depth-first search approach - in one test, the PGA search took 18.94 seconds while DFS took 9972 seconds. [15]

Abbasian and Mazloom took an almost identical approach to Rahnavard and Dastghaibyfard. The encoding process of chromosomes is the same as digits are assigned to numbers and empty letters are used for the remaining digits. The fitness function once again calculates how close the assigned digits are to a desired solution and mutations switch the assignments as well. Additional methods for parallelization are discussed also - it is mentioned that three parameters should be used to calculate how many individuals should be passed along to each thread. The number of threads, the internal population size, and the number of chosen individuals should also all be considered. Finally, the results of running this PGA algorithm against a regular depth-first search are shown in a graph which indicates that as the number of variables in the cryptarithmetic problem increase, PGA performs 1000 times better. [1]

Finally, Minhaz and Singh also describe a potential PGA solution for the cryptarithmetic problem by describing how a PGA search's efficiency will greatly depend on the way a chromosome is represented, how mutations and crossovers occur, and the accuracy of the fitness function. The use of an array of bits to encode the chromosomes is also discussed. This approach is similar to the digit assignment corresponding to each letter setup from the previously mentioned studies. Unlike the approaches discussed previously, this one uses a crossover technique to construct new offspring. A crossover spot is created in the array of bits and the array is crossed over along this axis to create new offspring chromosomes. The results show once again that the PGA approach is several hundred times faster than a traditional depth-first search approach. In addition, it is demonstrated that the memory

requirement of a PGA search is over half as low as a DFS search. [12]

3.3 Wordle

An article by Martin Short discusses two methods to efficiently determine a solution for Wordle.[17] The article first mentions the use of a most rapid decrease (MRD) algorithm. This is a greedy algorithm that simply aims to shrink the number of possible guesses. This means that MRD does not consider any step beyond the one that it is currently on. Short also mentions that a greatest expected probability algorithm can be used. This algorithm attempts to choose a guess that will lead to the greatest expected probability (GEP) that the correct solution will be chosen for the next guess. It should also be noted that both algorithms determined that the word TARES is the optimal starting word for the game. Short also concludes that in normal mode (where the next guess does not have to match the hints from every previous guess) MRD takes more rounds to win but it loses less frequently than the GEP algorithm. In hard mode, GEP prevails over MRD as the algorithm is able to utilize the hard mode rules to its advantage.

Another approach to efficiently solving Wordle is discussed in a paper by Bahinting et al.[3] The study details a slightly different game than Wordle as it uses four-letter words instead of five. The researchers used letter frequency to build a heuristic to rank each possible valid guess. The possible guesses are stored in a direct acyclic graph, where each edge of the graph is a letter and each vertex has a maximum of one outgoing edge for each letter. This data structure allows for the elimination of both prefix and suffix redundancies as the common prefixes/suffixes are shared between the strings that represent word guesses. This was noted to be a major memory usage reduction when storing the possible guesses. Overall, the combination of the letter frequency rankings and the clues gained from each guess permitted an efficient, in terms of runtime, technique to find the correct solution. It was also noted that when compared to a human player, the number of guesses taken to find the correct solution was reduced.

Brown proposes an active learning approach to solving Wordle in his paper. Using a Bayesian optimization formalization of active learning, Brown lays forward an approach one can take to solve Wordle using AI. First, the parameter space is defined (all the possible word guesses), then a surrogate model is built (to see how likely a word is to be correct) and is updated every time a guess is made given the new knowledge we learn about the letters. Brown gives three strategies for giving an evaluation/score to a guess - assigning a score based on how common the letters are, prioritizing words that allow the player to eliminate the most words, and valuing words that give the most information. The second two strategies resulted in a win more than 90% of the time. In addition, Brown notes that active learning has helped show that guessing words with the most information gain (rather than words that strictly are guesses close to the solution) results in a 99.7% success rate. Balancing exploration and exploitation is incredibly important in these kinds of problems. Brown's main argument for solving Wordle is that it is extremely important to define a word's value and how to use that to explore the parameter space. [5]

Finally, Bhambri et. al. discuss a reinforcement learning approach to solving Wordle, or

more specifically, a Partially Observable Markov Decision Process (POMDP) using a form of online search. Similar to Brown's paper, this one also stresses the importance of choosing guesses that provide the most information gain rather than one that simply tries to guess the correct word. However, this strategy is slightly changed as a rollout approach is used. This consists of a base heuristic that determines a word to guess given information from before and then uses Q-factors to improve upon it. Similarly to Short, Bhambri et. al. analyze the use of MRD and GEP heuristics but focus on the MIG (maximum information gain) heuristic. Using this MIG heuristic along with a rollout approach, a near-optimal solution is obtained. By setting up a belief state, this process can be modeled as a POMDP problem because there are unknown factors influencing what the decisions will be (in this case, how good of a guess the previous word was). By using rollout with the MIG heuristic and the POMDP model, the article details that this approach is near optimal and performs much better and more efficiently than traditional heuristic approaches. [4]

3.4 Literature Conclusions

While Wordle itself is a relatively new game and research on the use of artificial intelligence to help solve it is limited, the general concept behind the game and how it is played is not new and has been explored in other games. Two such games that have been extensively researched are Mastermind and Cryptarithmetic. The use of evolutionary/genetic algorithms in both of these scenarios could indicate that this might be a good approach for Wordle. However, the nonrandom structure of words, in general, does indicate that evolutionary/genetic algorithms may not provide the most efficient approach to solving Wordle. Rather, the currently available research on Wordle shows the use of various heuristics and data structures that are quite similar to those used throughout the Mastermind and Cryptharithmetic puzzles. In addition, in all of these approaches, the common consensus appears to be to use a strategy that maximizes information gain.

4 Solution Approach

The state was defined using an approach similar to one discussed by Andrew Ho who represented the Wordle state in Python using a dictionary with each letter in the English alphabet as a key. The value associated with each key consisted of a tuple storing whether or not that letter had been guessed, as well as a string of possible locations for that letter. The number 1 indicates that the letter can appear in that spot, the number 2 means that the letter must appear in that spot, and the number 0 means that the letter cannot appear in that spot.[8] As an example, the grid demonstrated below is formed after guessing the word "ADIEU" as the first guess, where the actual word is "ALOUD":

State Representation After Guessing “ADIEU” as the First Word:

A: 1: 21111
B: 0: 01111
C: 0: 01111
D: 1: 00111
E: 1: 00000
F: 0: 01111
G: 0: 01111
H: 0: 01111
I: 1: 00000
J: 0: 01111
K: 0: 01111
L: 0: 01111
M: 0: 01111
N: 0: 01111
O: 0: 01111
P: 0: 01111
Q: 0: 01111
R: 0: 01111
S: 0: 01111
T: 0: 01111
U: 1: 01110
V: 0: 01111
W: 0: 01111
X: 0: 01111
Y: 0: 01111
Z: 0: 01111

In this scenario, the letter ‘A’ appears in the first slot in the actual word, so it is marked as a 2 in the state and the value of the first index in all of the other lists is set to 0. This is because it is known that no other letters will appear in that slot. Then, the lists corresponding to the remaining letters that have been guessed will be updated. ‘E’ and ‘I’ both don’t appear in the final word, so their strings are made up entirely of zeros, and there likely is no longer a need to guess words with those letters. The implementation used in this study was similar, except the value of each entry in the dictionary was simply a list of length 6 composed of integers where the first position contained whether the word had been guessed or not and the remaining positions detailed the location of the letter in the solution word.

There were also nine algorithms developed in this study that were evaluated on their effectiveness in solving and winning Wordle. It should be noted that after each guess, the dictionary of possible words is pruned using the information gained. These algorithms are detailed below:

- RandomGuesser (RG)
  - Randomly select words to guess based on the dictionary of possible words.
- HighPositionRankGuesser (HPRG)
  - Rank each word in the dictionary of possible words, giving each letter in the word a score and summing these values. If the letter is duplicated in the word, only use the first occurrence. Each letter in the correct spot will be attributed a 2, a letter that has the possibility of being in that spot a 1, and a letter in the incorrect position 0. Then choose the word with the highest score (best ranking) and use that for the guess.
- LowPositionRankGuesser (LPRG)
  - Use the same ranking system as the HPRG algorithm, but for the second guess, use the word with the lowest score before switching to guessing the word with the highest position rank.
- AveragePositionRankGuesser (APRG)
  - Use the same ranking system as the HPRG algorithm but use a word with an average score for the second and third guesses before switching to guessing the word with the highest position rank.
- TwoPresetWordGuesser (TPWG)
  - Use the same ranking system as the HPRG algorithm but use two preset words for the first and second guesses to guess 10 different letters. Then switch to guessing the words with the highest position rank.
- EntropyRankGuesser (ERG)
  - Determine the ranking of each word by first determining the frequency of each letter in the dictionary of possible words and dividing this value by the length of the dictionary to determine a letter frequency probability value (x). Afterward, compute the entropy of each word by summing the entropy of each letter in the word using the following equation:

    $$\text{LetterEntropy} = x * (1 - x)$$

    Then, guess the word with the highest entropy value.
- HybridEntropyPositionGuesser (HEPG)
  - Use the same ranking system as the HPRG and ERG algorithms by finding the average of the normalized rankings for each word. Then, guess the words with the highest position rank.

- HybridEntropyTwoPresetGuesser (HETPG)
  - Use the same ranking system as the TPWG and ERG algorithms by first guessing the two preset words and then guess the words with the highest entropy ranking.
- MaxPruneGuesser (MPG)
  - Find the top 10 words based on the position ranking used in the HPRG algorithm and test each of those words against every possible solution word to calculate the average number of words pruned considering the feedback that would be received from each possible solution. Guess the word with the highest average number of words pruned, i.e. the word that provides the most information gain.

5 Experiment Design and Results

The Wordle game used in the testing of the algorithms was developed by Michele Pratusevich. [14] The algorithms discussed in this paper were incorporated to directly run with this code. The code is built to run in-terminal, so there is no GUI involved. It is a relatively simple implementation - it simply contains functions that simulate a playthrough of the game. Functions from Pratusevich's implementation are used in the experiment to create the word lists, validate guess words, and compare guess words to the actual word to return a string representing the feedback for making that guess. This feedback is then processed and used to update the state space designed for this experiment.

As mentioned previously, there are two lists of words that Wordle uses for its potential words. One list is a list of possible guess words - that is, all the valid words that can be guessed by the user. There is another list of "game" words that the answer for each game could be, which is approximately 1/6 the size. Every algorithm in this paper guesses words from the guess words list, and all the possible answers are within the game words list. The lists used were obtained from a GitHub repository authored by Finn Künkele who gathered the data directly from the official Wordle source code.[9]

With the lists available to the program, the state space and word dictionary can be built. The state space and its representation are described in section 4. In addition, it is also important to discuss how the word dictionary was built. For every algorithm except RG, varying ranking systems of unguessed and possible words were used. For this to occur, a Python dictionary of words and their corresponding score was created. A function called createRankingDict() was designed to do exactly this, iterating through the entire list of guessable words and assigning each word a score of 0 to start. Because the first guess of every algorithm is the word TARES, the ranking dictionary will immediately be updated with the response from guessing TARES, meaning when each algorithm runs, there will be scores that are nonzero in the word dictionary (unless none of the letters in TARES appear in the solution word).

As mentioned in section 4, the word dictionary is pruned after each guess. This process involves updating the state space to show the possible positions of each letter in the alphabet. If a given letter has a 0 in any of its positions, it means that that letter cannot be in that

position in the game word. This logic is used to prune words after each guess. Every word remaining in the word dictionary is examined to determine if any of its letters have a 0 in their corresponding positions in the game word. If a letter does have a 0 in its position, the word is pruned because it is certain that this will not be the game word. This pruning algorithm greatly improves the run time by eliminating a large number of words after each guess, hence shrinking the word dictionary substantially.

As was mentioned before, the initial guess used for all algorithms was the word TARES, used by Martin Short in his solution implementation.[17] This choice was demonstrated to provide the greatest information gain on average. It makes logical sense because it is a word containing very common English letters, including the very common vowels "a" and "e". This word is simply hard-coded in as the first guess for all algorithms. This decision resulted in the TPWG algorithm consistently choosing the word LINUM as the second guess.

Each algorithm, other than MPG, was tested over 2000 game iterations in order to ensure that the data was accurate. MPG was tested for 500 iterations due to the long run time. Algorithms were tested on a Windows 10 PC with an AMD Ryzen 5 3600 processor and 16 GB DDR4 CL 16 3200 MHz RAM.

It should also be noted that each algorithm was assessed using three different factors:

1. Average number of guesses needed to guess the solution word
2. Percentage of games won (solution word guessed in six or fewer guesses)
3. Time per game

The first two factors are quite similar in that a lower average number of guesses will often result in a high percentage of games won and so these two factors were weighed the highest when evaluating the algorithms. It should be mentioned, however, that the percentage of games won should be the primary factor when assessing the algorithms as a loss, no matter by how much, is still a loss. On the other hand, the time factor was simply used to determine whether the implementation of the algorithm was experimentally and practically viable. This allowed for the creation of a cumulative score. For each algorithm, the highest score was used to normalize the metric values between 0 and 1 and each of the above three assessment factors was weighted (70% for games won, 25% for average guesses, and 5% for time efficiency). These values were then summed to determine the ranking of each algorithm, where a lower score indicated a "better" algorithm. Results are listed below in Tables 1 and 2.

Table 1. Wordle Solution Algorithm Results

Algorithm	Average Guesses (#)	Games Won (%)	Time (sec/game)
RG	5.853	71.0%	0.02482
HPRG	4.676	91.4%	0.02901
LPRG	5.050	87.2%	0.03037
APRG	4.777	90.0%	0.02708
TPWG	4.560	93.8%	0.03123
ERG	6.010	66.3%	0.03316
HEPG	5.000	86.7%	0.03589
HETPG	4.821	92.4%	0.03783
MPG	4.804	92.2%	44.00

This table displays the performance of each algorithm in regard to the three performance measures described above. Algorithms and their implementation are described in section 4.

Table 2. Wordle Solution Algorithm Ranking and Overall Score

Rank	Algorithm	Overall Score
1	TPWG	0.7851
2	HPRG	0.7920
3	APRG	0.8000
4	HETPG	0.8112
5	HEPG	0.8172
6	LPRG	0.8243
7	MPG	0.8592
8	RG	0.8740
9	ERG	0.8794

This table displays the final ranking of all algorithms tested, ranked by lowest overall score (a lower score indicates a "better" algorithm), a value calculated from the three performance measures described above.

6 Discussion

The nine algorithms listed in Tables 1 and 2 were analyzed based on the average number of guesses it took to win each game, the game win percentage, and the runtime per playthrough. While the ultimate goal of each algorithm should be to win as many games as possible, in line with the rules of Wordle itself, evaluating these other statistics provides some further information useful to determine which algorithm is the most practical and useful. In addition, it is important to once again note that all algorithms used the word TARES as the first guess. Furthermore, it was noted that the average guess metric provided a simple

method to compare the results in this study to others. It was noted that when comparing solution implementations that used concepts such as reinforcement learning, the average guess count was approximately 3.9 guesses.[8] While this value is significantly smaller than those demonstrated in Table 1, concepts such as reinforcement learning and machine learning were not used in the algorithms implemented in this study. Rather, these algorithms were designed to solely use statistical methods to determine the next best guess in a game of Wordle for simplicity purposes.

In Table 2, an overall score column is included for each algorithm, aggregated based on the performance of that algorithm in the other three metrics. The weights of each metric were simply determined by converting a qualitative breakdown of each one of the three factors into a quantitative score, so the rankings demonstrated in the table may not be perfectly in agreement with similar literature implementations of each algorithm, however, it still serves to provide information necessary to make clear conclusions on the effectiveness of each algorithm in solving Wordle.

It should be noted that while the average number of guesses per game and the percentage of games won are related, they are not always directly correlated. As an example, the LPRG algorithm had a higher number of average guesses than the HEPG algorithm but also had a higher percentage of games won. The same relationship can be observed between the HPRG and HETPG algorithms. Therefore, it is prudent to consider both of these statistics when analyzing the results of the study.

The algorithms built in this study aimed to take key concepts from solution implementations for similar games such as Mastermind and Cryptarithmetic (as discussed in section 3). In regard to Mastermind, literature approaches often differentiated exploration and exploitation phases (such as the TPOA study authored by Chen et al.), a theme that the TPWG and HETPG algorithms attempted to implement.[6] This is because these algorithms guessed two preset words (the exploration phase), before aiming to directly guess the solution word (exploitation). While the implementation and overall concept may not be perfectly the same between this study and the literature implementations, the general idea did permit effective algorithms to be developed. As for cryptarithmetic, the general nature of constraints used in this puzzle is quite similar to the constraints that are used in the algorithms to prune the possible word dictionary. For example, the positional ranking algorithms in this study use constraints to evaluate where a letter can be in the solution word. This allows the algorithms to score each possible guess word, allowing for words to be selected in accordance with the information gained from the previous guesses.

Furthermore, as discussed in section 3.3, there are studies specific to Wordle itself that detail the use of algorithms that utilize letter frequency, such as the implementation described by Bahinting et al.[3] A similar approach was also used in the algorithms in this study, as exemplified by the HEPG algorithm which uses an entropy ranking system that is based on letter frequency. Other studies also used forms of online search, which simply describe an agent that takes an action (such as guessing a word), observes its environment afterward (the result of the action), and then takes an action using the previously gained information. This is quite a general definition of this type of search, but one that can be consistently seen

in the algorithms used in this study. This is because each algorithm aims to learn from the result of a guess and use that information to formulate the next guess.

The RG algorithm employs a strategy of guessing a completely random (but valid according to the result information) word each game. While this algorithm was not the worst one employed (based on the overall ranking), it had the second-to-worst average guess count and win percentage. This algorithm was simply used as a benchmark to compare other algorithms and won 71% of its games played. The rest of the algorithms tested all used a ranking system of some kind. The HPRG algorithm used a ranking system that gave priority to words that had guessed letters in spots that are/could be correct (described in further detail in section 4). This algorithm also selected its guesses by finding the word in the ranked word dictionary that had the highest rank. Its performance is notable for such a simple algorithm. It had the second-lowest number of average guesses and the fourth-highest win percentage. Clearly, it performed much better than the RG algorithm. This is due to the fact that like other algorithms, it iteratively approaches the solution word, using both the information it learns from the guesses as well as an approach that decreases the number of possible guess words.

The LPRG algorithm, using the same ranking as HPRG, also performed better than the RG algorithm. However, the results indicate that guessing the lowest-ranked word for the second guess does not result in a higher information gain than guessing the highest-ranked word in the dictionary. In addition, the APRG algorithm picks the word with an average score in comparison to the other words in the ranking dictionary. From the results, it can be concluded that guessing a word at this ranking provides more information gain for the second and third guess than the LPRG algorithm does, but less than the HPRG, which performed better in all three metrics seen in Table 1.

The TPWG algorithm was the best-performing algorithm overall. It had the lowest number of average guesses and the most games won. In this algorithm, two preset words with no common letters between them were guessed to start out each game. This method was noted to prune a significant percentage of words in the guess word dictionary, increasing the information gained, and allowing for the solution word to be guessed in fewer attempts than other algorithms. Literature sources corroborated this conclusion, discussing the fact that the optimal strategy was typically to guess two preset words at the beginning to prune as many words as possible before switching to guessing optimal words based on some ranking system.[13] This is a core conclusion determined through this experiment - the fact that guessing two preset words at the beginning consistently produces excellent results.

The ERG algorithm, on the other hand, produced the worst scores in every category (except runtime). Determining a guess solely based on the entropy ranking evidently did not result in a viable algorithm for solving Wordle. The HEPG algorithm performed markedly better than the ERG algorithm but was still worse than most of the other algorithms tested. By using a hybrid ranking system using both entropy and the positional word rank, performance improves significantly, but ultimately this algorithm still falls short of other algorithms tested in this study. Based on the results, it can be concluded that while entropy ranking does demonstrate improvement over the baseline RG algorithm, the position ranking system

is preferred due to its better performance.

The HETPG algorithm improves upon the HEPG algorithm by employing the same strategy as for TPWG - guessing two preset words at the beginning to try to prune the word dictionary as much as possible. This strategy, based on the data, was determined to greatly improve the performance of the algorithms that it is paired with. The HETPG algorithm has the fifth lowest average guess number but the second highest win percentage as 92.4% of games were won with this strategy.

All algorithms discussed thus far have had minimal differences in runtime, as they are all within the same magnitude of time. However, the MPG algorithm has a very poor runtime on the machine it was tested on, having a time per game that was approximately greater than the other algorithms by a magnitude of 10^3 which greatly reduces its practicality. However, it had the fourth-lowest average guess count and third-highest win percentage. This is because it is a simplified $O(n^3)$ algorithm. It looks at the top 10 highest-ranked words in the dictionary (rather than every word, due to impractical runtimes) and compares each of those words to every other word left in the possible words dictionary to see which word would, on average, prune the largest amount of words from the dictionary. Although its runtime is far higher in comparison to the other algorithms, it should be noted that its performance is ranked quite well in other categories. However, the overall score, along with the runtime, makes it difficult to conclude that this is the most efficient algorithm to use.

7 Conclusion

This paper analyzed the performance of several different algorithms attempting to solve the popular online game Wordle. After reviewing previous literature and formalizing the state representation for this search problem, nine different algorithms were built and experimented upon and their results were compiled. Upon compiling and reviewing results, it was found that the algorithms using a positional ranking system performed the best. In addition, it was noted that starting the search with two preset guesses that do not share any common words between them resulted in a performance that was "better" than all the other tested algorithms.

Despite the high win percentages and lower average guess count for some of the algorithms tested in this paper, there are still several ways to improve upon the algorithms discussed. The literature reviewed before starting the implementation of these algorithms revealed several solutions that resulted in approximately a 99% win percentage. Clearly, the algorithms discussed in this paper can be improved such that almost any possible target word can be guessed within 6 guesses. Much of the reviewed literature revolves around machine learning techniques to solve Wordle. This topic was outside the scope of the topic of this paper, but upon further research and learning of these methods, a better algorithm using some kind of machine learning could be implemented to attain better performance.

In addition, reviewing and refining the algorithms discussed in this paper could yield better results as well. For example, one of the main conclusions drawn in this study was that guessing two preset words from the beginning seems to increase the performance of multiple

different algorithms. Using this logic, then, adding this feature to the MPG algorithm could improve its performance in all categories, especially its efficiency. This is because after making the first two guesses, the word dictionary will be pruned significantly, meaning that the algorithm does not have to search through as many words when finding the average amount of words pruned. With more time and resources, ideas like this one would be pursued further.

The link to the GitHub repository containing the code for each algorithm is provided here: https://github.com/DIWAN005/WordleSolver

8 Author Contributions

- Mikkel Folting: Wrote sections 2, 3, 4, 5, 6, 7
- Safwan Diwan: Wrote sections 1, 2, 3, 4, 5, 6

References

[1] Reza Abbasian and Masoud Mazloom. Solving cryptarithmetic problems using parallel genetic algorithm. In *2009 Second International Conference on Computer and Electrical Engineering*, volume 1, pages 308–312, 2009.

[2] Benton J. Anderson and Jesse G. Meyer. Finding the optimal human strategy for wordle using maximum correct letter probabilities and reinforcement learning, 2022.

[3] J. L. Bahinting, J. D. G. Bonos, C. J. A. Delfinado, M. M. Ignacio, and L. F. Lachica. Computer-aided word mastermind solving using regular expression pattern matching in a directed acyclic word graph. *International Journal of Future Computer and Communication*, 2(5):481–484, Oct 2013.

[4] Siddhant Bhambri, Amrita Bhattacharjee, and Dimitri Bertsekas. Reinforcement learning methods for wordle: A pomdp/adaptive control approach, 11 2022.

[5] Keith A. Brown. Model, guess, check: Wordle as a primer on active learning for materials research. *npj Computational Materials*, 5 2022.

[6] Shan-Tai Chen, Shun-Shii Lin, and Li-Te Huang. A two-phase optimization algorithm for mastermind. *The Computer Journal*, 50(4):435–443, 2007.

[7] Shan-Tai Chen, Shun-Shii Lin, Li-Te Huang, and Sheng-Hsuan Hsu. Strategy optimization for deductive games. *European Journal of Operational Research*, 183(2):757–766, 2007.

[8] Andrew Ho. Solving wordle with reinforcement learning, Feb 2022.

[9] Finn Kunkele. Wordle competition. `https://github.com/Kinkelin/WordleCompetition/tree/main/data/official`, Feb 2022.

[10] Roel Leus, Lotte Berghman, and Dries Goossens. Efficient solutions for mastermind using genetic algorithms. *Computers & Operations Research*, 36(6):1880–1885, 2009.

[11] J.J. Merelo-Guervós, P. Castillo, and V.M. Rivas. Finding a needle in a haystack using hints and evolutionary computation: the case of evolutionary mastermind. *Applied Soft Computing*, 6(2):170–179, 2006.

[12] Aadil Minhaz and Ajay Vikram Singh. Solution of a classical cryptarithmetic problem by using parallel genetic algorithm. In *Proceedings of 3rd International Conference on Reliability, Infocom Technologies and Optimization*, pages 1–5, 10 2014.

[13] Tom Neill. Ruining the fun: a wordle auto-solver. `https://notfunatparties.substack.com/p/wordle-solver`, Jan 2022.

[14] Michele Pratusevich. A python wordle clone. `https://www.practicepython.org/blog/2022/02/12/wordle.html`, Feb 2022.

[15] Gholamali Rahnavard and Gholamhossein Dastghaibyfard. An efficient parallel algorithm for solving cryptarithmetic problems: Pga. In *2009 Third UKSim European Symposium on Computer Modeling and Simulation*, pages 102–106, 2009.

[16] Anu Sharma, Nitin K. Verma, and K.B. Anand. Breadth first search & genetic algorithm as an effective technique to solve crypt arithmetic problems. *Resurging India - Myths and Realities*, pages 95–99, 2012.

[17] Martin B. Short. Winning wordle wisely—or how to ruin a fun little internet game with math. *The Mathematical Intelligencer*, 44(3):227–237, 2022.