AI Search and Natural Language

Pavel Tokarev

May 2025

1 Abstract

This project compares three different implementations for AI search strategies to solve Wordle. Rule-Based, Frequency-Based, and Entropy-Based algorithms. All methods achieved a 100% success rate on a random sample list of 500 valid Wordle guesses. The Entropy-Based solution had the fewest average attempts per word, while having the longest execution time for the total test, compared to the Rule-Based and Frequency-Based implementations. The results highlight the trade offs between average attempts and computational cost, and their potential use cases.

2 Introduction

Word games have long served as tools for entertainment and a way to improve one's cognitive abilities. To be effective players in these games, individuals would have to use and develop their skills in language, pattern recognition, and strategic reasoning. Historically classic games such as Scrabble, Hangman, and Word Search have been used to test one's brain power and vocabulary in competitive and educational settings, socially determining a winner whose intelligence is "superior". As the public's frame of view changes to the advanced developing technology being introduced, interest has instead shifted to looking at and developing Artificial Intelligence that can outperform human performance.

One of the most popular, recently developed word games is Wordle. It is a five-letter word guessing game that gives feedback after every guess, allowing a maximum of five attempts per game. This creates an interesting dilemma for AI development. Unlike games such as tic-tac-toe or chess, Wordle provides only a partially observable world with great uncertainty and constraints. This creates an ideal situation in order to evaluate multiple algorithmic search approaches to determine optimal strategies for word games. The purpose of this paper is to analyze the theoretical and practical applications of research on AI algorithm strategies for word games by looking at the previous academic reports that have been done as it relates to this area, particularly highlighting Rule-Based Systems, Heuristic Search, Entropy Strategies, and other previous work done in word games.

3 Literature Review

3.1 Foundations of Rule-Based Systems

The concept of a rule-based system for an AI program is a concept that has been around for decades. Essentially, it is a rule-based reasoning approach in which a decision-making algorithm set by a designer follows a set of predetermined rules to reach some sort of conclusion. As defined by Davis & King in 1984, "one side of a rule is evaluated with reference to the database, and if this succeeds (i.e., evaluates to TRUE in some sense), the action specified by the other side is performed."[3] Essentially, a rule-based reasoning system is a structured system of inquiries leading down a specific path to reach a finished result. This algorithm aims to mimic human logic and reasoning, an important step in the development of autonomous artificial intelligence. Because of its relatively transparent system, it provides highly interpretable steps and results. Davis & King were pioneers in the use of the rule based algorithm, developing an early system for giving medical diagnoses called MYCIN[3]. Based on the symptoms the patient was presenting with, it could infer the illness. Another example of this technique being used for autonomous problem solving was DENDRAL, a rule-based system for chemical analysis[4]. The rules were set by an expert, creating an expert-crafted formalized domain of rules, simplifying and quickening the process. Through the use of this system, it would be possible to create a rule set of the domain of the English language to create a rule based system for word games. Through constraints set by a rule system, it would be possible to shrink the domain considerably after every move.

3.2 Greedy & Heuristic Search in Word Games

Greedy and heuristic algorithms have become essential in the development of Artificial Intelligence. They are especially useful when attempting to solve a problem where traversing large search spaces is required with a set of constraints. Strategies using letter probabilities played a significant impact in improving human performance [1]. Both human and AI strategies rely heavily on heuristics for decision-making [7]. A greedy algorithm makes decisions to search a space set on the option that gives the most immediate benefit, while a heuristic acts as a guide for choosing what option to take to get closer to the goal. In solving problems, like word games or in this case Wordle, this is particularly useful because of the limited feedback we receive after each guess. In order to efficiently solve these kinds of problems we must reduce the domain of the possible goal states. One common heuristic that has been used for Wordle involves selecting words with high-frequency letters early on, eliminating the most amount of letters in the domain. Studies have evaluated a number of these strategies by comparing the number of guesses and win rates, supporting that this method is more effective than other guessing models [1]. However, this strategy also has limitations despite its efficiency. Heuristics may lead to non-optimal paths if the underlying assumptions are wrong or too focused [6].

3.3 Entropy Based Strategies

Entropy-based search strategies are becoming an increasingly common approach in Artificial Intelligence decision-making. Based on information theory, entropy measures the uncertainty or information content in a set of outcomes [8]. When applied to problem-solving, it can be used to assign a value to how much information is gained by learning the value of an attribute; it measures the information gain of a decision. In the context of word games, such as Wordle, entropy can help evaluate how informative a guess might be in reducing the domain of possibilities. We can select guesses based on their expected information gain and choose the most informative word at each step. Previous research has explored the use of entropy in Wordle and specifically its effectiveness compared to heuristic approaches [5]. Similarly, entropy-based reasoning has been explored in large language models to improve performance in complex puzzles that provide limited feedback [9]. While effective, entropy-based methods can be computationally intensive, and in certain cases, straightforward frequency-based guessing can perform just as well or better [5]. By comparing the performance of an entropy-based search to other solutions, we can compare the improvement and cost in relation.

4 Approach: Representation and Algorithms

In order to explore different strategies discussed in the previous sections, an analysis can be done on simulating Wordle games with different implementations of AI search strategies. A common representation of the game state was created that allowed each "guesser" to access the game state and perform under identical conditions/constraints. The tools used to create these implementations were created using Python. The implementation consisted of a hidden five-letter word, and for each guess, the algorithm would receive feedback: g meant a correct letter and correct position, y meant a correct letter and wrong position, b meant a wrong letter. This feedback function was critical in order to allow algorithms to reduce the domain of possible words. The implementation of the feedback function is shown in Listing 1.

Listing 1: Feedback function

```
1  def get_feedback(guess, target):
2      feedback = ['b'] * 5
3      target_chars = list(target)
4      used_indices = []
5      for i in range(5):
6          if guess[i] == target[i]:
7              #...Assign letters in guess
8              # and target at the same position g...
9      for i in range(5):
10         if feedback[i] == 'b' and guess[i] in target_chars:
11             #...Assign rest of letters that are in the guess
12             # and target y, and rest b...
13     return ''.join(feedback)
```

The list was then updated to only hold words that were still possible to be the answer. Words are stored in a list and achieved from a list of all official Wordle answer possibilities, a smaller domain of all guessable words. This enforced the same rule set as Wordle, leaving words in which g letters match exactly, y are present, and g are excluded. The Rule-Based Algorithm relies only on elimination of the domain, the Frequnecy-Based will prioritize guesses with high frequency letters, and the Entropy-Based algorithm maximizes the expected information gain of a guess.

4.1 Rule-Based Algorithm

The basis of the Rule-Based algorithm is that it is simply an algorithm that follows a few steps to select a word without leveraging any heuristics, frequency, or probability. It fully relies on the feedback function in order to continually try words in a brute-force like method. The implementation is shown in Listing 2.

Listing 2: Rule Solver

```
1  def rule_based_solver(target, word_list):
2      guess = "slate"
3      attempts = 1
4      candidates = word_list.copy()
5      while guess != target:
6          feedback = get_feedback(guess, target)
7          new_candidates = []
8          for word in candidates:
9              if get_feedback(guess, word) == feedback:
10                 new_candidates.append(word)
11         candidates = new_candidates
12         if not candidates or attempts > 6:
13             return attempts, False
14         guess = candidates[0]
15         attempts += 1
16     return attempts, True
```

This algorithm starts with the initial guess “slate”, a common starter word for players, and eliminates all words that would produce feedback with a g. The next candidate words to be guessed is selected alphabetically from the list of remaining words, without the use of any heuristics or prioritization strategies. This repeats until either the guesser is out of attempts or the correct word is found.

4.2 Greedy/Heuristic Algorithm

The goal of this algorithm is to reduce the search space quickly and efficiently using letter frequency. The frequency of letters is calculated by iterating through the word list and calculating the most frequently appearing letters in a specific position. The implementation is shown in Listing 3.

Listing 3: Frequency list builder

```
def letter_frequency(words):
    freq = [Counter() for _ in range(5)]
    for word in words:
        for i, char in enumerate(word):
            freq[i][char] += 1
    return freq
```

This creates a frequency word list that can be used to determine the most common letters in the domain. Before the main algorithm creates a guess, it first ranks each word according to how many most common letters it has. The implementation is shown in Listing 4.

Listing 4: Frequency table builder

```
def score_word(word, freq_table):
    score = 0
    seen = set()
    for i, char in enumerate(word):
        if char not in seen:
            score += freq_table[i][char]
            seen.add(char)
    return score
```

This will allow the algorithm to choose the word with the greatest number of the most frequent letters in it. The implementation for the main algorithm is shown in Listing 5.

Listing 5: Frequency Solver

```
def frequency_based_solver(target, word_list):
    possible_words = word_list[:]
    attempts = 0
    while attempts < 6:
        freq_table = letter_frequency(possible_words)
        scored_words = [(word, score_word(word, freq_table))
        for word in possible_words]
        scored_words.sort(key=lambda x: x[1], reverse=True)
        guess = scored_words[0][0]
        feedback = get_feedback(guess, target)
        attempts += 1
        if feedback == 'g' * 5:
            return attempts, True
        possible_words = update_possible_words(possible_words, guess, feedback)

    return attempts, False
```

This will sort words by the largest score, given by the amount of frequent letters a word contains. So it is then choosing the highest probability word for each guess.

4.3 Entropy-Based Algorithm

The Entropy-Based algorithm uses concepts from Information Theory, discussed previously in the literature section. It uses information gain in order to guide the decision on what word should be guessed. The algorithm measures the expected information gain from each potential guess, based on how much it reduces uncertainty among the remaining words. Ones that yield a better information gain, or in other words reduce uncertainty are seen as better guesses. This should yield a effective strategy for choosing guesses.

In order to evaluate each guess, the algorithm calculates the entropy of each possible word it could guess afterwards with the feedback it receives. It uses all the words that would remain as possible candidates to calculate the probabilities, and in return calculate the entropy using the formula:

$$I(guess) = -\sum_{w \in W} P(f) \log_2 P(f)$$

Where W is the set of all possible feedback from the remaining domain of words, and $P(f)$is the probability of receiving a specific feedback for the possible guess f, which is the feedback for a particular word. This is implementation can be seen in Listing 6.

Listing 6: Entropy Calculator

```
1  def calculate_entropy(guess, possible_words):
2      feedback_counts = defaultdict(int)
3      for word in possible_words:
4          feedback = tuple(get_feedback(guess, word))
5          feedback_counts[feedback] += 1
6      total = len(possible_words)
7      entropy = 0.0
8      for count in feedback_counts.values():
9          p = count / total
10         entropy -= p * math.log2(p)
11     return entropy
```

This creates a dictionary that will count the number of times each feedback pattern occurs. Then it receives feedback for every possible word that is still inside the domain, and takes a count of how many times each feedback pattern occurs. Then it is possible to calculate the probability of receiving a specific feedback pattern which is used to calculate the entropy. Its goal is to maximize the reduction of uncertainty with every possible guess against every possible word that could be the solution. Or, in other words, it calculates the entropy of a guess by simulating the feedback pattern with every possible remaining word. This function is called by main algorithm which returns the results for every single possible word and picks the word with the greatest information gain.

Listing 7: Entropy Solver

```
1  def entropy_based_solver(word_list, target_word, precomputed_entropy=None):
2      possible_words = word_list.copy()
```

```
    attempts = 0
    success = False
    while attempts < 6:
        if attempts == 0 and precomputed_entropy is not None:
            entropy_scores = precomputed_entropy
        else:
            entropy_scores = [(word, calculate_entropy(word, possible_words))
            for word in possible_words]
            entropy_scores.sort(key=lambda x: x[1], reverse=True)
        guess = entropy_scores[0][0]
        feedback = get_feedback(guess, target_word)
        print(f"Guess: {guess}, Feedback: {feedback}")
        attempts += 1
        if guess == target_word:
            success = True
            break
        possible_words = update_possible_words(possible_words, guess, feedback)
    return attempts, success
```

It is important to note that this implementation of a Wordle solver is particularly much more computationally expensive. Making exponentially more calculations than the other algorithms.

5 Experiment & Results

The experiment was set up with a word list of official guessable 5 letter words in Wordle.[2] This totaled 2315 words. Due to resource complications, a 500 random word sample was taken from the total list. The results from running the three different algorithms, show the important balance between efficiency and computational cost. A summary of the results is shown in Table 1.

Algorithm	Avg. Attempts	Success Rate	Execution Time (s)
Rule-Based	3.16	100%	0.51
Frequency-Based	3.09	100%	1.19
Entropy-Based	2.98	100%	309.70

Table 1: Performance metrics

The Average Attempts represent the average number of attempts it took each algorithm to successfully find the answer for each word. The Success Rate tracks how many words each algorithm could successfully find. The Execution Time represents the number of seconds it took to complete all 500 guess cases.

6 Analysis

All of the algorithms achieved a 100% success rate with the constraints placed by Wordle, showing that each approach was able to solve the Wordle game under the same constraints and receiving the same feedback. The differing metric between the three algorithms was Average Attempts & Execution Time.

In terms of Average Attempts, the Entropy-Based solver outperformed both Frequency-Based and Rule-Based with an average of 2.98 attempts per word. The other two algorithms had an average attempt number of 3.09 and 3.16, respectively. However in terms of Execution Time, it was the complete opposite. The Entropy-Based solver required 309.7 seconds, while the Frequency-Based took 1.19 seconds, and the Rule-Based took 0.51 seconds. The reason that such a large amount of time for the execution of the Entropy-Based algorithm was required was because of the large number of calculations required for every possible guess against every possible answer in every iteration. This exponentially increased the complexity of this implementation. The Frequency-Based and Rule-Based implementations still achieved a 100% success rate while maintaining a very fast execution time.

These results showcase the trade offs between Execution Time & Avg. Attempts. Therefore, suggesting that the Entropy-Based algorithm provided the lowest average attempts needed to solve this word game. However, the other algorithms performed similarly, with a much faster execution time.

7 Conclusion

This project aimed to explore three different AI search strategies for solving Wordle. These three approaches included a Rule-Based system, a Frequency-Based System, and an Entropy-Based system. All three of these algorithms were successful in achieving a 100% success rate when ran with a random sample of 500 words from the official valid Wordle answers list. The Entropy-Based solution ended up having the best average guess performance, with an average of 2.98 guesses per word, with the cost of a significantly higher runtime of over 300 seconds to solve 500 words. However, both the Frequency-Based and Rule-Based solvers also provided a 100% success rate with a runtime of under 1.5 seconds for the entire 500 word list, making them more appropriate options for systems where there are resources and or time constraints. The full 2315 list could not be used due to the resource constraints of Google Colab when evaluating the Entropy-Based model.

Future work on this exploration could focus on a better optimization of the Entropy-Based solver approach in order to reduce its computational demand and runtime, allowing for a better demonstration on the full word list. Another possible direction could be testing the performance of other search algorithms such as A*, or the performance of natural language processing agents with word games. Additionally, allowing for a less constrained list of 5 letter words may lead to differing performances. Hybrid approaches could be another

intriguing route to take this exploration, combining different heuristics and strategies to finding the best balance between Avg. Attempts and execution time.

References

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

[2] cfreshman. wordle-answers-alphabetical, 2022.

[3] Randall Davis and Jonathan J King. The origin of rule-based systems in ai. *Rule-based expert systems: The MYCIN experiments of the Stanford Heuristic Programming Project*, 1984.

[4] Edward Feigenbaum, Bruce Buchanan, and Joshua Lederberg. On generality and problem solving: A case study using the dendral program. *Machine Intelligence*, 6, 09 1970.

[5] Chao-Lin Liu. Using wordle for learning to design and compare strategies. In *2022 IEEE Conference on Games (CoG)*, pages 465–472, 2022.

[6] Norvig Peter and Russell Stuart Artificial Intelligence. *A Modern Approach*. Pearson Education, USA, 2021.

[7] Matīss Rikters and Sanita Reinsone. Strategic insights in human and large language model tactics at word guessing games, 2024.

[8] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[9] Graham Todd, Tim Merino, Sam Earle, and Julian Togelius. Missed connections: Lateral thinking puzzles for large language models, 2024.

8 Notes:

The full Python Notebook, with all the helper functions, can be accessed here:
https://github.com/Ptokarev74/WordleAIExploration
Entropy-Based solution Inspired by:
https://github.com/GillesVandewiele/Wordle-Bot
List of valid Wordle solutions:
https://gist.github.com/cfreshman/a03ef2cba789d8cf00c08f767e0fad7b