Jennifer Wang (xw2763)

Project Proposal: Parallel Wordle

Overview

This project aims to build a parallel solver for NYT's popular Wordle game[1], where players get six tries to guess a 5-letter word. After each try, the Wordle game responds with one of the three colors for each letter of the player's guess: green means the letter exists in the answer and is at the correct position; yellow means the letter exists in the answer but is not at the correct position; grey means the word does not exist in the answer. A player wins if they guess the word correctly on their 6th try and loses otherwise.

Wordle Solver

I will explore three different algorithms for solving the Wordle game: frequency, entropy, and minimax. I will then implement parallel strategies for the algorithm with the best performance. If time permits, I will also parallelize the other two algorithms.

Solver Algorithm 1: Frequency

This is the most basic algorithm for a Wordle solver: first loop over all available words in the dictionary, then count the frequency of each character to create a letter-to-frequency map. Finally, normalize the map so that the sum of all character frequencies is 1. The Wordle solver will proceed to selecting the word with the highest frequency sum, normalized by the number of repeated characters in the word.

Solver Algorithm 2: Entropy

When a player makes a guess, there are 3^5 = 243 possible responses from the Wordle game (three outcomes for each letter and 5 letters for each word). For example, "2 yellows, followed by 1 grey, followed by 2 greens" is one possible response. We can visualize these 243 responses as a histogram with 243 buckets, one for each of the possible responses.

Intuitively, we want our next guess to fall into the bucket with the least frequency, as it prunes the most of our search space. The extreme case is the all-green-response bucket: the frequency is 1 and we will immediately win. On the other hand, we don't want our next guess to land in a bucket with a large frequency. In order to avoid the worse-case scenario and strive for the best-case scenario, we can't rely purely on luck. Our strategy is therefore to choose a word that leads to a distribution with the most uniform distribution, so that we get a guaranteed reduction of 1/N of the search space. It won't be the best, and it won't be the worst. It will be average.

We can measure the uniformity of a distribution via the following entropy function $H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$. We favor words with higher entropy because a greater entropy indicates a greater uniformity.

Solver Algorithm 3: Minimax

This algorithm is inspired by the game Absurdle[2]. In essence, when selecting the next best word, we always want to assume that the Wordle Game gives us the "hardest" word. "Hard" in this context means the word that prunes the least amount of search space (maximizes the remaining search space). The wordle solver will therefore anticipate this situation. From all the valid words, it would choose a word that minimizes the remaining search word from the anticipated next word given by the wordle game.

I will use alpha-beta pruning for reducing the search space during each iteration. I will also limit my search depth to 2 (one round of minimize and one round of maximize).

---

[1] https://www.nytimes.com/games/wordle/index.html

[2] https://qntm.org/files/absurdle/absurdle.html

Data
The NYT Wordle uses two datasets. It has a 2,315 common five-letter words dataset that it uses for answer-selection. It also has a 10,657 less-common five-letter words dataset that dictates the possible words that a player can input into Wordle. For this project, I will be using both datasets to in my algorithms. In theory, any dataset containing 5-letter words would work.

Wordle Player
The Wordle player will be implemented in an automatic fashion. This is so that we can evaluate the quality of the three Worle algorithms. It will loop through all 2,315 valid five-letter words to stimulate 2,315 round of Wordle games. At each round, the automatic Wordle player will use one of the solver-algorithms above to guess a word, and after obtaining a response from the Wordle game, it will once again use that algorithm to obtain its next guess. The game terminates when the wordle player guesses the answer.
If time permits, I will implement an interactive version of the Wordle player.

Algorithm Quality Measurement
Two questions are asked to evaluate the quality of the three Wordle solver-algorithms:

1. *Will it win*: can the algorithm solve all 2,315 rounds of Wordle in six attempts?
2. *Will it win skillfully*: what's the average number of attempts for solving all 2,315 rounds of Wordle?

Question #1 will take precedence over question #2 during the evaluation process.

Parallelization
There are two main parts that we can parallelize. The first part is to parallelize the automatic wordle player so that the 2,315 rounds of Wordle are played in parallel. This is crucial for speeding up the evaluation process.
The second part is to parallelize the wordle solver. I will start with the algorithm with the best performance. If time permits, I will parallelize the other two algorithms.

References

- Frequency: https://arxiv.org/pdf/2202.03457.pdf
- Differential Entropy: https://aditya-sengupta.github.io/coding/2022/01/13/wordle.html
- Minimax:
    - inspiration (Absurdle): https://qntm.org/challenge
    - Algorithm: http://aima.cs.berkeley.edu
- Wordle Dataset: https://www.kaggle.com/datasets/bcruise/wordle-valid-words?select=valid_guesses.csv