Exploring Wordle Puzzles using Evolutionary Algorithms

Anushree Ganesh, Ella Has, and Richard Schwartzkopf
Group 15

June 14, 2024

1 Introduction

Wordle is a web-based word game that was developed during the COVID-19 pandemic and gained popularity in 2021 through the New York Times. The game involves a five-letter target word, and the objective for players is to guess this word within six attempts. Players receive feedback based on the color of the tiles: yellow indicates that the guessed letter is in the target word but in a different position, while green signifies that the guessed letter is in the correct position. Additionally, grey-colored tiles indicate that the letter is not part of the target word. See Figure 1 for an example of a Wordle game being solved in four guesses.

Given the game's large search space and unique feedback mechanisms, we sought to explore an algorithm capable of incorporating these characteristics. Consequently, Evolutionary Algorithms (EAs) were chosen for this purpose. Evolutionary algorithms are employed to estimate solutions to complex optimization problems and are often tailored to meet the specific requirements of a particular problem, as no single algorithm can competitively solve all problems [6].

1.1 Problem Statement

The main problem statement is: Can Evolutionary Algorithms solve Wordle-like puzzles and what are the effects of different mutation strategies and fitness functions on its efficiency? We compared the performance of genetic algorithms (GAs) with different mutation strategies such as constant mutation, deterministically changing adaptation etc. and different fitness functions where feedback from green letters or sum of green and yellow letters etc. were taken into consideration.

1.2 Review of related work

A small selection of previous literature has informed the experimental setup of this project. Evolutionary algorithms have been applied to solve Wordle puzzles. In [4], a genetic algorithm was used to generate guesses for solving the Wordle puzzle. To this end, they applied a selection criterion to find the most fit members of a given population of words to generate guesses. Mutation, inversion and permutation was applied to the selected members to generate "children". After a fixed number of generations, the word with the highest fitness was used as a guess. Additionally, other papers have also tried using GAs to solve Wordle such as [3] which used an Agent's algorithm coupled with a genetic component as well as a Minmax component.

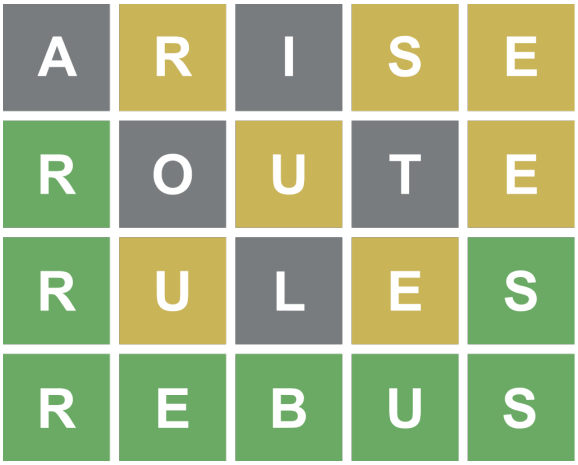

Figure 1: An example of how to play Wordle. Source: Wordle Fandom

Other relevant papers to our question involve how to tune the choices made for a genetic algorithm. In [5], different static and adaptive mutation techniques are compared on a range of problems. We used this as a source for adaptive mutation rate strategies, and an inspiration for how to set up our experiments. Two adaptive mutation strategies compared in this paper, a deterministically changing rate [1] and a rate changing based on the fitness spread of the population [7] were particularly interesting because of their simple implementation, explained further in the Methods section.

2 Methods

2.1 Evolutionary Algorithm Ingredients

As our baseline algorithm, we wrote a generic form of a genetic algorithm (GA) which initializes a population of words randomly selected from the provided word list. For all tests, the population size was 10. This was chosen to allow for crossover, while limiting how long it takes to compute one generation.

2.1.1 Fitness Functions

Game feedback (green, yellow, and grey), indicating the correctness of individual letters and their positions, was incorporated in different fitness functions, which are used to evaluate how close each guessed word is to the target word. Three different fitness functions were created:

- **Basic Fitness Function**:
  - Score is based on correct letter at correct position (green letters)
- **Sum Fitness Function**:

  - Score is based on the number of fully correct (green) and correct but in the wrong place (yellow) letters.

- **Weighted Fitness Function**:
  - Assigns different weights to correct letters in the correct positions (green matches) and correct letters in the wrong positions (yellow matches).
  - Weights are chosen arbitrarily as 5 for green letters and 1 for yellow letters. Green letters are better to have, and so they are weighted more.

2.1.2 Parent Selection

The parents, which are used to create new words, are selected based on their fitness scores using a proportional selection strategy, in which the parent selection probability is proportional to the fitness scores. During crossover two pairs of selected parents generate the offspring, with the crossover point being randomly chosen to get some variability.

2.1.3 Mutation

Two different ways to mutate the words were implemented. The first way was the basic mutation of swapping a character for a random character from the alphabet. All characters were equally weighted to keep it simple. This way of mutating the word could not use the information given by yellow letters, that the letter is in the word but not in the position it is in, because letters are never moved within the word. For that purpose, a second mutation strategy was implemented: swapping mutation. When a character is mutated, it has a probability proportional to the number of yellow letters in the word to be swapped out for a character from the word itself instead of the full alphabet. Due to a substantially higher chance of swapping a character with itself (1/5 compared to 1/26 in the case of 5-letter words), it was also tested whether it makes a difference to avoid such mutations which are in effect not a real mutation.

- **Basic Mutation**:
  - Each character has a chance of the mutation rate $(1/L)$ of being exchanged for any letter from the alphabet.
  - This does not allow for use of the yellow letter information.

- **Swapping Mutation**:
  - Each character has a chance of the mutation rate $(1/L)$ of being mutated.
  - If a character is mutated, it has a chance of the number of yellow letters divided by the total length of the word of being swapped for a letter in the word. Otherwise it is swapped for a letter in the alphabet.
  - There is a higher chance of choosing the same character again, so in one variation, choosing the same character is avoided by removing it from the list of options.

2.1.4 Adaptive Mutation Rates

Finally, also the mutation rate was varied. Generally, evolutionary algorithms have a mutation rate of $1/L$ due to its simplicity and yet high performance [2]. However, it may be beneficial to adapt the mutation rate throughout the generations. We explore three such mutation adaptation strategies and compare it to the constant mutation rate of $1/L$.

- **Deterministic**

  The deterministic approach uses the formula in Equation 1 to determine the mutation rate at each generation [1]. Here, γ is the mutation rate, L is the length of the word, g is the current generation and N_g is the maximum number of generations. This mutation rate gradually increases over the generations.

$$\gamma = \frac{1}{L}(1 + \frac{g(L-1)}{N_g}) \tag{1}$$

- **Number of correct letters**

  A second strategy is based on the number of correct letters in the best word in the current population. This rate is designed to ensure that the expected number of mutated letters in a word is 1. Since green letters are never mutated, the number of mutable letters reduces with increasing number of green letters, and so to mutate one letter, the chance of mutation must increase with the number of green letters. It is thus defined as Equation 2, where γ is the mutation rate, L is the length of the word, f_{max} is the maximum fitness in the current population.

$$\gamma = \frac{1}{L - f_{max}} \tag{2}$$

- **Fitness distribution**

  The final strategy is based on characteristics of the fitness distribution [7], as shown in Equation 3, where γ is again the mutation rate, γ_0 is the initial mutation rate (in this paper set to $1/L$), β and κ are coefficient factors (both set to 1), and f_{min}, f_{max}, and f_{avg} are the minimum, maximum, and average of the fitness of the population respectively. With this adaptation, the mutation rate decreases when the distance between the best and worst words is larger. The average fitness only has a small impact, but the mutation rate does increase slightly with increased average fitness.

$$\gamma = \gamma_0 \left(1 + \beta \frac{(f_{avg})^\kappa}{(f_{max} - f_{min})^\kappa + (f_{avg})^\kappa}\right) \tag{3}$$

2.2 Motivation for testing methods

The experiments were done on 5-letter word lists. The word lists contained all solution words for Wordle. The performance of the GAs were compared with a brute-force approach that simply tested all words until the target word was found (without replacement) or sampled repeatedly from the word list (with replacement). For the experiments, nine different configurations were tested over N=100 trials each. The different combination where created by combining two mutation functions (mutate, mutate_swap, and mutate_swap with exclude_self being true) with three fitness functions (calculate_fitness_any_position, calculate_fitness_sum_match, and calculate_fitness_weighted_match).

The genetic algorithm was executed over multiple generations. The maximum number of generations was set to a high number to ensure that the target word will always be found. For each experiment time was measured because it is the most robust indicator of how well the algorithm scales and could be used to compare the brute-force approach to the GA approach. The number of individuals the algorithm has seen was tracked to measure how extensively the search space has been explored before arriving at a solution. By tracking the number of generations, we could investigate how the interactions of different fitness functions and mutation strategies affected convergence time.

3 Results

3.1 Mutation Strategies and Fitness Functions

Variations on the baseline algorithm were made with regards to the fitness function and the way mutation was implemented. All combinations between these were tested on time and number of individuals. Looking at Figure 2, it seems that the most basic mutation strategy, which exchanges a letter for any letter from the alphabet, performs best across all fitness functions, both in terms of seen individuals and run time.

The performance of swapping mutation, where a letter is exchanged for a letter within the word, with a chance proportional to the number of yellow letters in the word, is somewhat improved by forbidding exchange for itself. This matches our expectation, because exchanging a letter for itself is a waste of time and is much more likely to occur when picking from the word than when picking from the alphabet.

Compared to the brute force algorithms, the evolutionary algorithms perform similarly in terms of individuals seen. The brute force algorithm without replacement, which simply goes through the word list in order, is much faster, but with its memory of its guesses, it is a high standard to put on the evolutionary algorithms. The brute force algorithm with replacement is a lot slower than any of the algorithms.

3.2 Adaptive Mutation Rates

Another aspect of the baseline algorithm that was explored was the mutation rate. The run times and number of individuals can be seen in Figure 3. A constant mutation rate of $1/L$, where L is the length of the word, was compared with three adaptive mutation rate strategies. The deterministic mutation rate change did not seem to make much of a difference. Perhaps this is due to the fact that it uses the maximum number of generations allowed, which was set quite high to ensure convergence, and so in the generations it took to find the word, the mutation rate was not changed much.

The mutation rate based on the number of green letters in the best word does take less long to converge. Considering that this strategy aims to have an expected number of mutated letters per word of 1, this is in effect an application of the preferred mutation rate of $1/L$ from the literature for the specific application where L changes over generations. It is thus to be expected that this would work well, given the general consensus on the effectiveness of $1/L$.

Finally, the mutation rate adaptation from [7] that uses characteristics of the fitness distribution of the population performs well too, especially looking at the number of individuals.

All evolutionary strategies outperform the brute force algorithms in terms of individuals. The brute force strategy without replacement is again very fast, but as mentioned before, looking through a list is very efficient and a harsh standard for a search algorithm.

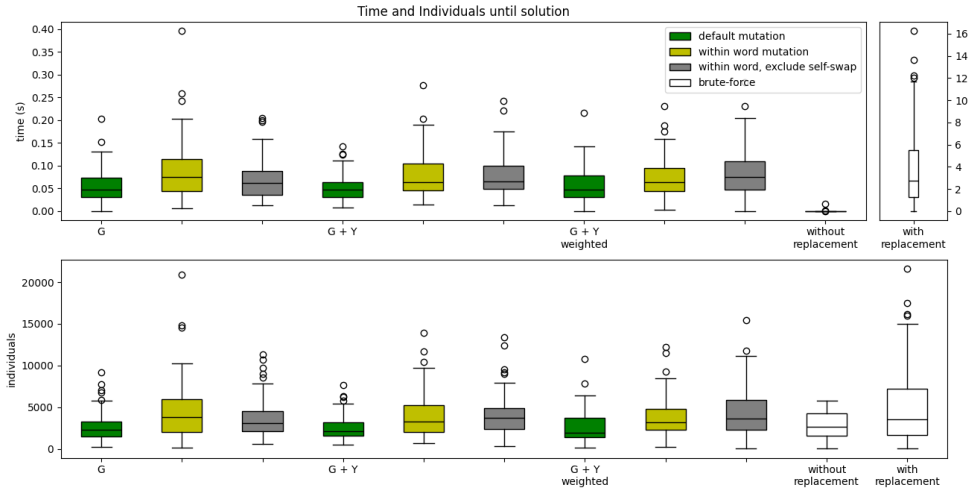

Figure 2: Distribution of time taken (top) and number of individuals seen (bottom) until a solution was reached for 100 words per combination of fitness function (G: green only, G+Y: green + yellow, G+Y weighted: 5 green + 1 yellow) and mutation strategy (default full alphabet, full alphabet and within word, full alphabet and within word excluding swapping with itself), as well as brute force algorithms sampling with and without replacement from the word list. Note the different time scale for the brute force with replacement.

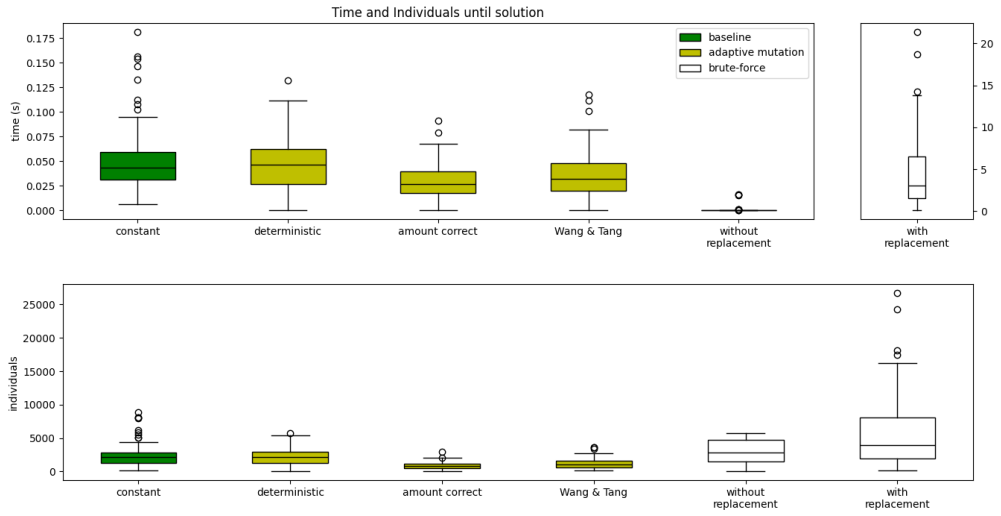

Figure 3: Distribution of time taken (top) and number of individuals seen (bottom) until a solution was reached for 100 words per mutation adaptation strategy (constant mutation, deterministically changing adaptation, based on amount of correct letters, based on fitness of the population (Wang & Tang, 2011)) and brute force algorithms sampling with and without replacement from the word list. Note the different time scale for the brute force with replacement.

3.3 Statistical Analysis

In order to test for statistical difference between the different types of fitness functions and the different mutation rates, a one-way analysis of variance (ANOVA) test was conducted. Since there were more than two independent groups (mutation strategies as well as fitness functions), an omnibus test was necessary to test whether the explained variance in the dataset was significantly greater than the unexplained variance. Hence, the F-statistic and its corresponding p-value was used. The null hypothesis tested is that there is no statistically significant difference in means of the different groups.

Independent one way ANOVA showed no significant effects of the type of mutation strategy (constant mutation, deterministically changing adaptation, Wang & Tang mutation, and Brute Force with or without replacement) on the time taken and the number of individuals seen until a solution was reached ($F\ (2) = 165.31$, $p>.001$) as reported in table 1.

Table 1: Analysis of Variance: The different mutation strategies have no significant difference in means

	Sum squared	Degrees of Freedom	F	p-Value (>F)
Mutation strategy	2.0378	2	165.31	1.254
Residual	1.1076	1797.0		

Independent one way ANOVA showed a significant effect of the type of fitness function (Green letters only, Sum of green and yellow letters, Weighted sum of green and yellow letters, and Brute Force with or without replacement) on the time taken and the number of individuals seen until a solution was reached ($F\ (2) = 1015.86$, $p<.001$) as reported in table 2.

Table 2: Analysis of Variance: The different fitness functions have a significant difference in means

	Sum squared	Degrees of Freedom	F	p-Value (>F)
Fitness function	7.6044	2	1015.86	0.0
Residual	1.234	3297.0		

Since, significant ANOVA results were found, Post-hoc tests were further conducted. The Bonferroni test was used since it is a conservative test and can check for guaranteed control over Type 1 error at the risk of reducing statistical power. The Tukey test was also conducted since it is commonly used to control Type 1 error for groups with same sample size and equal group variance. Post hoc testing using Tukey's correction revealed that the type of fitness function had significant effect on individual and times ($p<.001$). Further, the Bonferroni correction revealed that the group-wise comparison of means is statistically significant and the null-hypothesis can be rejected ($p<.001$) as reported in table 3.

Table 3: Post-hoc tests: The different fitness functions have a significant difference in means even after Tukey and Bonferroni corrections to control for type 1 errors. Multiple Comparison of Means - Tukey HSD, FWER =0.05

group1	group2	mean difference	p-adj	lower	upper	reject H_0
generations	individuals	2605.009	0.0	2411.573	2798.4288	True
generations	times	-995.3689	0.0	-1188.7968	-801.9409	True
individuals	times	-3600.3698	0.0	-3793.7977	-3406.9419	True

4 Discussion

The results provide interesting insights into the performance of different mutation strategies and fitness functions. One unexpected finding was that the basic mutation strategy consistently outperformed all other strategies even the more complex ones. This includes the swapping strategy, which took game feedback into account by taking the known positions of yellow letters into account. One explanation could be that broad exploration is more beneficial, than the more focused, but potentially constrained strategy of swapping letters within the word. This emphasizes the importance of maintaining a wide search scope especially in the early stages of the evolutionary process, because it allows the algorithm to explore many different possibilities before narrowing down to a more constrained solution space.

While the deterministic mutation rate change did not provide significant advantages, the mutation rate based on the number of green letters showed faster convergence. This confirms the efficacy of the $1/L$ mutation rate from the literature, adapted for our specific application. Additionally, the adaptive mutation rate from [7] performed well, especially in reducing the number of individuals needed to find a solution. However, its implementation could be optimized further. If fitness calculations were incorporated into parent selection and mutation rate adjustment, it would eliminate redundant computations and potentially enhance overall performance.

A notable limitation of our study is that the weights used in the fitness functions were set arbitrarily. Future work should design an algorithm that learns and adapts these weights over generations. This adaptive approach could potentially lead to more efficient convergence by fine-tuning the selection pressures based on ongoing results. Furthermore, it should be explored how the integration of adaptive mutation rates with other fitness functions and population sizes affects the fitness distribution characteristics as suggested by [7]. Additionally, incorporating weighted probabilities for picking letters from the alphabet, based on their occurrence frequency (e.g., 'E' being more likely than 'Q'), could improve the mutation strategies even further.

5 Conclusion

In conclusion, adding yellow letter information to the fitness function is difficult to exploit. We tried to allow for letters to move within the word, but it did not improve performance. What did seem to improve the run-time and number of generations needed, albeit not significantly, was adaptive mutation rates. Perhaps this can be exploited in the attempt to guess words while making use of the yellow-letter information. All in all, evolutionary strategies may not be the best performing algorithm for solving Wordle in the fewest guesses, but it is quick and simple to code, while find-

ing the word in less than a second without knowledge of the word list, with plenty of space for improvement.

6 Code

The code can be found here.

7 Contributions

7.1 Anushree Ganesh

- Coded the brute force algorithm
- Conducted the statistical analysis on the data acquired from tests
- Wrote the introduction section
- Wrote the statistical analysis subsection of the results

7.2 Ella Has

- Coded the mutation adaptation strategies in `mutation.py`.
- Coded the test loops and visualisations and ran the tests.
- Wrote the user guide.
- Wrote the methods parts about mutation types and mutation adaptation.
- Wrote the results section.
- Wrote the conclusion.

7.3 Richard Schwartzkopf

- Coded the baseline GA.
- Coded the fitness functions.
- Wrote the methods parts about the baseline and the fitness functions.
- Wrote the discussion.

References

[1] Sunith Bandaru, Rupesh Tulshyan, and Kalyanmoy Deb. "Modified SBX and adaptive mutation for real world single objective optimization". In: *2011 IEEE Congress of Evolutionary Computation (CEC)*. IEEE. 2011, pp. 1335–1342.

[2] Á. E. Eiben, R. Hinterding, and Z. Michalewicz. "Parameter control in evolutionary algorithms." In: *IEEE Transactions on evolutionary computation* 3(2) (1999), pp. 124–141.

[3] G Matthews. "Solving Wordle Using Artificial Intelligence". In: *GitHib Repository* (2023).

[4] D Nichols. "Wordle: Finding the Right Words to Say". In: *GitHib Repository* (2022).

[5] B. R. Rajakumar. "Static and adaptive mutation techniques for genetic algorithm: a systematic comparative analysis." In: 8(2) (2013), pp. 180–193.

[6] D. Shilane et al. "a general framework for statistical performance comparison of evolutionary computation algorithms". In: *Information Sciences* 178 (14 2008), pp. 2870–2879. DOI: 10.1016/j.ins.2008.03.007.

[7] Lei Wang and Dun-bing Tang. "An improved adaptive genetic algorithm based on hormone modulation mechanism for job-shop scheduling problem". In: *Expert Systems with Applications* (2011).