WordleR Performance Analysis (aka 'The Autoplayer')

John S. Erickson

2024-05-30

The Challenge: Are there any Wordle puzzles that WordleR can't find?

There are many possible "challenge" word lists to test the WordleR algorithm against, including Knuth (5757 words) and Kaggle (over 39k). Ultimately we found the Wordle "Magic Words" list (2315 words).

Our default setup for this notebook is to load the Wordle Magic Word list, *re-arranged by scoring each word by the frequency of its letters with respect to the entire list* and de-emphasizing duplicate letters. "For completeness" other word list options are available in the code comments for testing and analysis, and a few utility code snippets are embedded to help with hacking when needed.

Initialize the "challenge word" list: 39k words (Kaggle), 5757 words (Knuth), 2315 words (Wordle) or Wordle Remaining Words

The "challenge words" comprise the possible words this notebook's Wordle emulator has available to use. In the development of WordleR we tested the algorithm with five-letter word lists from Kaggle and Knuth before settling on the 2315-word Wordle list, obtained through various sources.

Here we read in the selected word list.

```
# Initialize our word list: 39k words (Kaggle), 5757 words (Knuth), 2315 words (Wordle) or 1829 words (
#short_list.df <- readRDS("unigram_freq.Rds") # Kaggle's 39K words, sorted by frequency
#short_list.df <- readRDS("short_list.Rds") # Knuth's 5757 words, sorted by letter frequency
#short_list.df <- readRDS("Wordle_Words.Rds") # DEFAULT: "Official" Wordle Magic Words, by letter freq
#short_list.df <- readRDS("used_words_full.Rds") # Full list of Wordle magic words

# 23 Mar 2024: Only use remaining Wordle words (from WordleR)
used_words.df <- readRDS("used_words.df.Rds")
short_list.df <- readRDS("Wordle_Words.Rds") # Official Wordle Words, sorted by their word frequency sc
short_list.df <- anti_join(short_list.df, used_words.df, by="word")

short_list.df <- short_list.df %>% select(word) # Only need 'word' column
```

```
# select the top n words by frequency from the selected word list
n <- nrow(short_list.df)

# make it a vector
short_list <- short_list.df[1:n,]$word
```

Create a starter word list, for analysis

We test a mixture of words recommended by various sources:

- The most frequent English word (ABOUT)
- Words with three or four vowels

- A few "Best Wordle Starter Words" from several online sources: "YouTubing PhD students," Prof. Horstmeyer, et.al.
- Exhaustively-tested starter words; only "winning" words used now (23 Mar 2024)

```
# Recommended starter words from the Slate article: https://slate.com/technology/2022/02/perfect-wordle
# trial_words <- c("stern","rents","nerts","terns",
#                  "slart","certs","crest","tyler",
#                  "larnt","styre")

# Recommended start words from Horstmeyer et.al. https://bit.ly/3Koycju
# # fastest
# trial_words <- c("slice","tried","crane","leant",
#                  "close","trice","train","slate",
#                  "lance","trace","stern")

# # most likely to succeed
# trial_words <- c("adept","clamp","plaid","scalp",
#                  "clasp","depot","print","recap",
#                  "strap","tramp","stern","start",
#                  "equal","style","crane")

# # Best of Horstmeyer's, olyerickson's and NYTimes words
# trial_words <- c("recap","dealt", "adept","clasp","start",
#                  "stern","strap","clamp","equal")

# # NEW: The old recommended starter words have all been used!
# trial_words <- c("heart","recap", "haste","stare",
#                  "stone","earth","tenor","irate")

# Testing NYTimes' WordleBot Starter Word: 11 Apr 2022
# trial_words <- c("recap","dealt")

# Testing Martin B. Short's words: 24 Mar 2022
# trial_words <- c("tares","rates","tales","aloes","saner","roles","lanes","riles","tears","rotes")


# # Starter words from RPIrates (09 Feb 2022)
# trial_words <- c("equal","bayou","stare","query",
#                  "adieu","proxy","about","crane",
#                  "stern","slart","crest","tyler",
#                  "larnt","styre")

# 23 Mar 2024: Long run!
#trial_words <- short_list.df$word[101:150]

# 23 Mar 2024: Long run, only winners!
# winners_0150 <- readRDS("winners_0150.Rds")
# winners_51100 <- readRDS("winners_51100.Rds")
# winners_101150 <- readRDS("winners_101150.Rds")
# winners_151200 <- readRDS("winners_151200.Rds")
# winners_201300 <- readRDS("winners_201300.Rds")
# winners_301400 <- readRDS("winners_301400.Rds")
# winners_401500 <- readRDS("winners_401500.Rds")
# winners_001200 <- readRDS("winners_001200.Rds")
```

```r
# winners_5011000 <- readRDS("winners_5011000.Rds")
# winners_001500 <- readRDS("winners_001500.Rds")
# winners_0011000 <- rbind(winners_001500,winners_5011000)
# saveRDS(winners_0011000,"winners_0011000.Rds")
# winners_001500 <- readRDS("winners_001500.Rds")
# trial_words.df <- winners_001500 %>%
#   select(starter) %>%
#   rename(word = starter)
#
# trial_words <- unlist(lapply(trial_words.df$word, as.character))

# Based on winners 1-500
# trial_words <- c("fetid", "demon", "haunt", "evict",
#                  "clasp", "eying", "clove", "ladle",
#                  "ovary", "aware")
#
# Based on winners 1-1000
trial_words <- c("recap", "fetus", "flake",
                 "lyric", "ovary", "salvo",
                 "slang", "stork")
```

The Challenge

Overview: For each recommended WordleR "starter word," verify that any other word on the `short_list` can be reached by the WordleR "algorithm."

WordleR "Manual" Algorithm:

Assumes a word list (challenge words and potential guesses) pre-arranged by scoring the letter frequency of each letter of each word, and not scoring for multiple occurrences of a letter.

1. Select a starter word, for example one of recap, fetus, flake, lyric, ovary, salvo, slang, stork.
   - After many test runs, we usually just use **STERN** (Mar 2022)
2. Get Wordle's response:
   - Letters to exclude (grey squares)
   - Letters to include (yellow or green squares)
   - Letters to exclude by position (yellow)
   - Letters to include by position (green)
3. Modify the word list based on these results to reveal the remaining possible words. By pre-arranging the word list based on the frequency of each letter, we hope to increase the amount of "information" returned by Wordle for each of our guesses.
4. Select WordleR's top recommendation and submit to Wordle.

WordleR "Autoplay" Algorithm:

For each starter word from recap, fetus, flake, lyric, ovary, salvo, slang, stork:

1. The `starter` word is the first `test` word in the outer loop (could be manual)
2. Select the next `challenge` word from `short_list` (emulates Wordle's hidden word)
3. Evaluate `test` against the current `challenge`:
   - Letters to exclude: What letters aren't present at all?
   - Letters to include: What letters are present
   - Letters to exclude by position: What letters don't match, by position?
   - Letters to include by position: What letters are perfect matches, by position?
   - Update our data structure holding the results needed to guide our exclusion and inclusion steps.
4. Modify the word list based on these results to reveal the remaining possible words.

- By pre-arranging the word list based on the frequency of each letter, we hope to increase the amount of "information" returned by Wordle for each of our guesses.

5. Select the top word (first word in the remaining list) as the new test word
6. Repeat until `test` matches a member of `short_list` or the end of `short_list` is reached.

Exclusion/Inclusion Functions

These implement the WordleR exclude/exclude buttons...

```
# Filter a list of words based on a letter to exclude
exclude_letter <- function(letter,word_list) {
    exclude_mask <- !grepl(letter, word_list, fixed = TRUE)
    return(word_list[exclude_mask])
}

# Filter a list of words based on a letter to include
include_letter <- function(letter,word_list) {
    include_mask <- grepl(letter, word_list, fixed = TRUE)
    return(word_list[include_mask])
}

# Filter a list of words based on a letter to exclude in a certain position
exclude_letter_position <- function(letter,position,word_list){
    exclude_mask <- str_sub(word_list, position, position) != letter
    return(word_list[exclude_mask])
}

# Filter a list of words based on a letter to include in a certain position
include_letter_position <- function(letter,position,word_list) {
    include_mask <- str_sub(word_list, position, position) == letter
    return(word_list[include_mask])
}

# Filter a list of words based on a list of letters to exclude
exclude_letters <- function(letter_list,word_list) {
  result <- word_list
  for (letter in letter_list) {
    result <- exclude_letter(letter,result)
  }
  return(result)
}

# Filter a list of words based on a list of letters to include
include_letters <- function(letter_list,word_list) {
  result <- word_list
  for (letter in letter_list) {
    result <- include_letter(letter,result)
  }
  return(result)
}
```

Wordle Simulator "kernel"

This is *Wordle-as-a-function*; it returns a list containing the equivalent of Wordle's evaluation of our attempt, including letters to exclude or include, and their positions when appropriate

```
word_test <- function(test,challenge) {

  test <- strsplit(test,"")[[1]]
  challenge <- strsplit(challenge,"")[[1]]

  # letters in `test` that are not in `challenge`
  no_match <- test[!(test == challenge)]

  # letters in `test` that are also in `challenge`
  match <- test[(test == challenge)]

  # letter/position matching
  # These are perfect matches
  match_mask <- test == challenge  # TRUEs are "green"
  not_match_mask <- test != challenge  # TRUEs are "green"

  # What letters/positions should be "green"?
  green_match <- test[match_mask] # green letters
  green_match_pos <- which(match_mask)

  # What letters/positions should be "yellow"?
  yellow_match <- test[not_match_mask] # green letters
  yellow_match_pos <- which(not_match_mask)

  # The result is a list that we'll pick apart when creating our suggested next try
  result <- list(no_match,match,green_match,green_match_pos,yellow_match,yellow_match_pos)

  return(result)
}
```

Apply the WordleR algorithm

Here we apply the WordleR algorithm to the challenge word list, using our set of starter words.

- Test our guess (`attempt`) against the `challenge` word (the selected word from the list)
- Get the Wordle evaluation
- If the evaluation is a perfect match, exit!
- If not, use the evaluation to trim the list
- Refresh the `attempt` word; use the first word in our list of remaining possibilities
- Continue until we achieve a perfect match

```
# initialize our grand results structure
iteration_results <- data.frame(starter=character(),
                                challenge=character(),
                                iteration=integer())

result_list <- list()

# Test against a list of potential starter words.
# We use indices in case we want to parallelize later
for (i in 1:length(trial_words))  {
starter <- trial_words[i]

# k is the range of challenge words we test against
```

```
for (k in 1:length(short_list))  {
challenge <- short_list[k]

# Re-initialize for next challenge word
iteration <- as.integer(0)
word_list <- short_list
attempt <- starter  # Our first `attempt` is always our `starter` word

# Execute the WordleR algorithm
# `attempt` is the first entry in what remains of `word_list`
while ((length(word_list) != 0) && (!is.na(attempt) && (attempt != challenge))) {
  iteration <- as.integer(iteration + 1)

  # if (challenge=="watch") {
  # }

  # This is the Wordle evaluation!
  wordle_result <- word_test(attempt, challenge)

  # if (challenge=="watch") {
  #   print(attempt)
  #   print(wordle_result)
  # }

  # Here begins actual WordleR algorithm:
  # Exclude
  if (length(wordle_result[[1]])!=0)  {
    # Don't remove letters that are also includes
    # Use our own function to do the work!
     if (length(wordle_result[[2]])!=0){
       # Remove included letters from exclude list, if any
       exclude_list <- exclude_letters(wordle_result[[2]],wordle_result[[1]])
     } else {
       # Otherwise, the list is okay
       exclude_list <- wordle_result[[1]]
     }
    word_list <- exclude_letters(exclude_list,word_list)
  }

  # Include
  if (length(wordle_result[[2]])!=0){
    word_list <- include_letters(wordle_result[[2]],word_list)
  }

  # Include by position
  if (length(wordle_result[[3]])!=0){
    for (letter in wordle_result[[3]]){
      letter_pos <- which(strsplit(attempt,"")[[1]] == letter)[1]
      word_list <- include_letter_position(letter,letter_pos,word_list)
    }
  }

  # Exclude by position
```

```
  if (length(wordle_result[[5]])!=0){
    for (letter in wordle_result[[5]]){
      letter_pos <- which(strsplit(attempt,"")[[1]] == letter)[1]
      word_list <- exclude_letter_position(letter,letter_pos,word_list)
    }
  }

  # Choose the next attempt from the remains of `word_list`
  # NOTE: word_list has been pre-arranged by letter frequency score
  if (length(word_list) != 0) {
    attempt <- word_list[1] # The next guess
  } else {
  }
}

# Update iteration_results
iteration_results <- rbind(iteration_results,cbind(starter,challenge,iteration=iteration+1)) %>%
  mutate(iteration = as.integer(iteration))

} # end of challenge list
} # end of starter list

# factor-ize the starter word column, for pretty plotting later
iteration_results$starter <- as.factor(iteration_results$starter)
```

The Bottom Line!

```
# Aggregate the results

# Summarize the failures: starter words the required more than six iterations for certain challenge wor
iteration_results_misses <- iteration_results %>%
#  filter(starter=="bayou") %>% # In case we want results for one word
  filter(iteration > 6)

#iteration_results_misses
missed_words <- iteration_results[iteration_results$iteration > 6,]

# Save it for later
#write_csv(missed_words,"missed_words_1829.csv")
#write_csv(missed_words,"missed_words_2315.csv")
write_csv(missed_words,"missed_words_23Mar2024.csv")
#write_csv(missed_words,"missed_words_5757.csv") # The Knuth version

# What starter words "win" most often? (ie six iterations or less)
# Filter by iterations
iteration_results_wins <- iteration_results %>%
  filter(iteration <= 6)

iteration_results_summary <- iteration_results_wins %>%
  group_by(starter, as.factor(iteration)) %>%
  summarise(n=n())
```

```
## `summarise()` has grouped output by 'starter'. You can override using the
```

```
## `.groups` argument.
```

```
iteration_results_starter_success <- iteration_results_summary %>%
  group_by(starter) %>%
  summarise(total_challenge_wins = sum(n)) %>%
  arrange(desc(total_challenge_wins))
```

What is the 'Best' Wordle Starter Word?

Create a summary plot of the results using certain starter words.

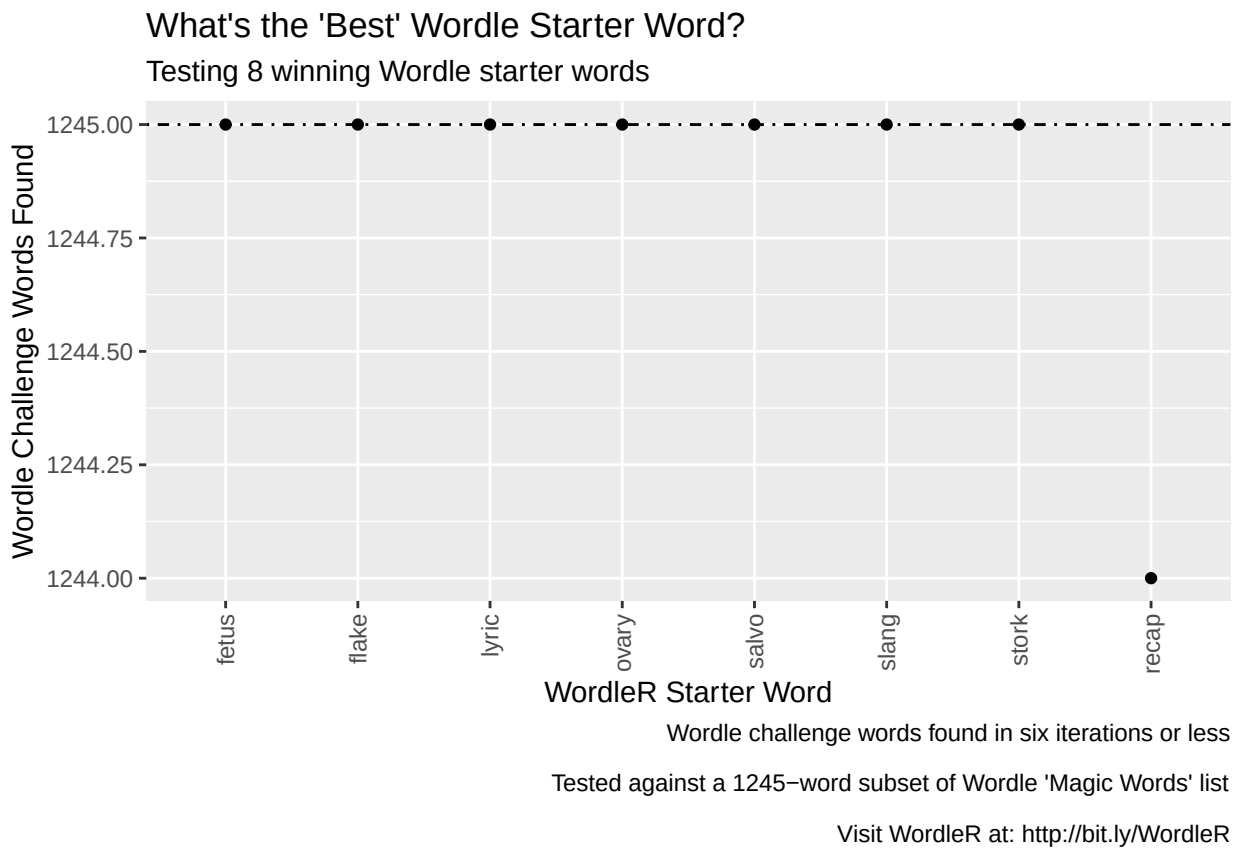
```
# Plot these results!
# Total words:
words <- nrow(short_list.df)

p1 <- ggplot(iteration_results_starter_success,
       aes(x=reorder(starter,-total_challenge_wins),y=total_challenge_wins)) +
  geom_point() +
  labs(
    title="What's the 'Best' Wordle Starter Word?",
#  subtitle = "Testing revised recommended starter word list: 18 Oct 2022",
#  subtitle = "Testing favorite Wordle starter words (including Horstmeyer and YouTuber) ",
  subtitle = paste0("Testing ", length(trial_words)," winning Wordle starter words"),
#  subtitle = "Testing Horstmeyer's & NYTimes words: 11 Apr 2022",
#    subtitle = "Wordle Challenge Words Found in Six Moves or Less vs Starter Words",
    caption = paste0("Wordle challenge words found in six iterations or less\n
                     Tested against a ",words,"-word subset of Wordle 'Magic Words' list\n
                     Visit WordleR at: http://bit.ly/WordleR")
    ) +
  xlab("WordleR Starter Word") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
  ylab("Wordle Challenge Words Found") +
  geom_hline(aes(yintercept = nrow(short_list.df)),linetype = "dotdash") +
  #geom_text(label=paste0("Wordle Magic Word List (truncated): ",words," words", x="equal", y=(words +
  NULL

ggsave(plot=p1,filename = "comparing_starter_words_winners_bar.png")
```

```
## Saving 6.5 x 4.5 in image
```

```
p1
```



All of the tested starter words will solve Wordle using a 1245 Magic Word subset.

More Plots Comparing Starter Words

Create a bar plot comparing the results using the 8 "winning" starter words.

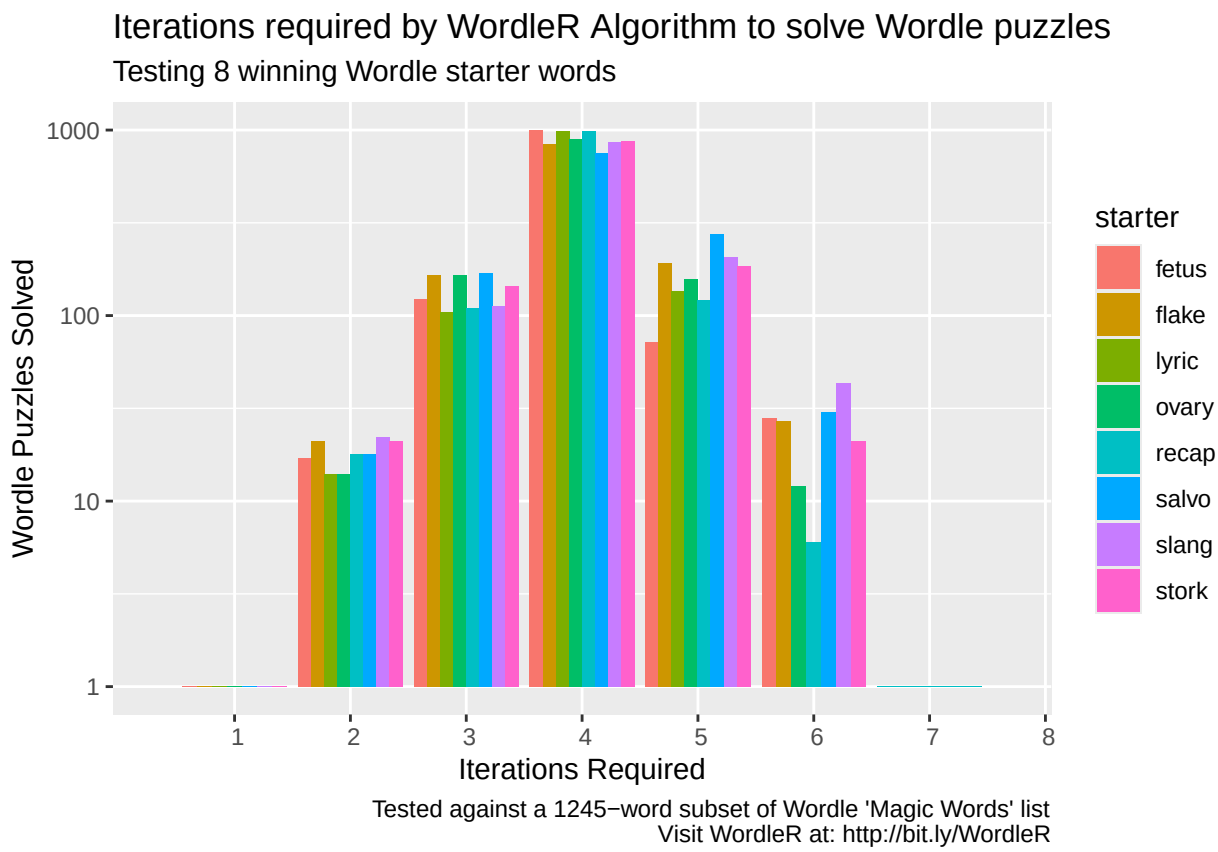
```
p2 <- iteration_results %>%
#  filter(starter=="bayou") %>%
  ggplot(aes(iteration, fill=starter)) +
  geom_bar(stat = "count",position = position_dodge()) +
  xlab("Iterations Required") +
  scale_x_discrete(limits=c(1,2,3,4,5,6,7,8,9,10)) +
  scale_y_continuous(trans='log10') +
  ylab("Wordle Puzzles Solved") +
  labs(
  title = "Iterations required by WordleR Algorithm to solve Wordle puzzles",
#  subtitle = "Testing NYTimes' recommended starter word: 11 Apr 2022",
#  subtitle = "Testing favorite Wordle starter words (including Horstmeyer and YouTuber) ",
#  subtitle = "Testing revised recommended starter word list: 18 Oct 2022",
  subtitle = paste0("Testing ", length(trial_words)," winning Wordle starter words"),
#  subtitle = "Testing favorite Wordle starter words (including Horstmeyer and YouTuber) ",
  caption = paste0("Tested against a ",words,"-word subset of Wordle 'Magic Words' list\nVisit WordleR a
```

```
## Warning in scale_x_discrete(limits = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)): Continuous limits supplied to
## i Did you mean `limits = factor(...)` or `scale_*_continuous()`?
```

```
ggsave(plot=p2,filename = "comparing_starter_words_iterations_bar.png")
```

```
## Saving 6.5 x 4.5 in image
```

```
p2
```



Create a line plot comparing the results using certain starter words.

```
line_size <- NA
line_size[1:length(trial_words)] <- 1
line_size <- data.frame(line_size)
line_size$starter <- trial_words
#line_size[line_size$starter=="heart",]$line_size <-  1.1 # Starter word to emphasize

p3 <- as.data.frame(iteration_results) %>%
  left_join(line_size) %>%
  ggplot(aes(iteration,color=starter)) +
  geom_density(stat = "count") +
  # geom_density(stat = "count",aes(size=line_size)) +
  xlab("Iterations Required") +
  scale_x_discrete(limits=c(1,2,3,4,5,6,7,8,9,10)) +
  scale_y_continuous(trans='log10') +
  ylab("Wordle Puzzles Solved") +
  labs(
  title = "Iterations required by WordleR Algorithm to solve Wordle puzzles",
#  subtitle = "Testing NYTimes' recommended starter word: 11 Apr 2022",
#  subtitle = "Testing Horstmeyer's & NYTimes words: 11 Apr 2022",
```

```
#  subtitle = "Testing revised recommended starter word list: 18 Oct 2022",
#  subtitle = "Testing favorite Wordle starter words (including Horstmeyer and YouTuber) ",
  subtitle = paste0("Testing ", length(trial_words)," winning Wordle starter words"),
  caption = paste0("Tested against a ",words,"-word subset of Wordle 'Magic Words' list\nVisit WordleR a
  geom_vline(xintercept = 6)
```

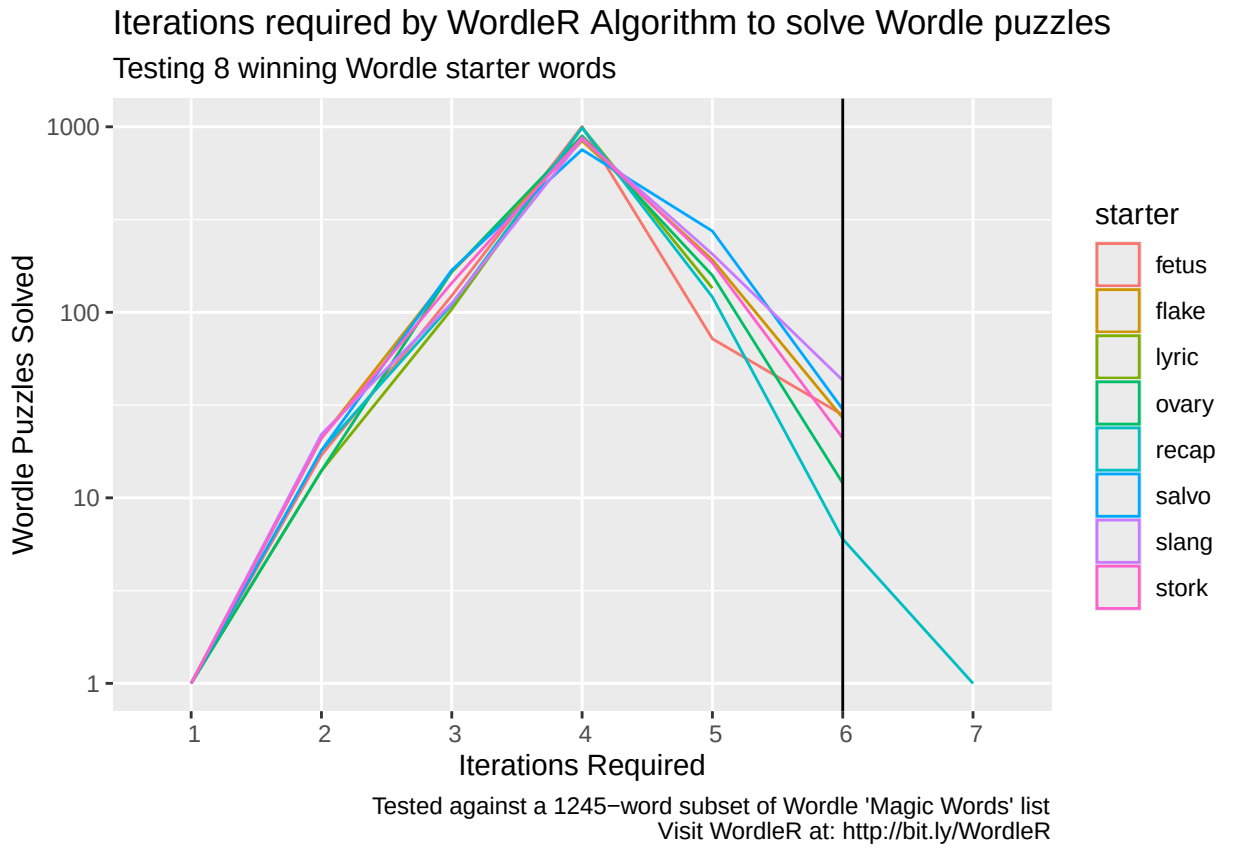
```
## Joining with `by = join_by(starter)`
```

```
## Warning in scale_x_discrete(limits = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)): Continuous limits supplied to
## i Did you mean `limits = factor(...)` or `scale_*_continuous()`?
```

```
ggsave(plot=p3,filename = "comparing_starter_words_iterations_line.png")
```

```
## Saving 6.5 x 4.5 in image
```

```
p3
```



What Winning Starter Word Will Survive the Longest?

Of the 8 "winning" starter words tested, what word can we use the longest?

In theory, whether a starter word is among the remaining challenge words doesn't actually matter; a proven "winning" starter will work for the duration of Wordle (based on testing against the original Magic Words list). But it is interesting to ponder, what word from our "winners" list will be the last to disappear from the Magic Words list?

We can figure this out by evaluating the position of our 8 in the 1245 Magic Words subset.

```r
# First figure out where the trial words appear
positions <- match(trial_words,short_list)

# Then find the word in the highest position
last_standing <- short_list[max(na.omit(positions))]

last_standing
```

```
## [1] "ovary"
```