

The best strategies for Wordle, part 2

By Alex Selby, 17 March 2022.

Contents

The Best Strategies for Wordle, part 2

Updated best starting words using New York Times word lists as of 30 August 2022

Updated best starting words using New York Times word lists as of 16 March 2022

What makes a good strategy from a human point of view?

Analysis of your play

Choosing two words at a time

Using larger hidden word lists

Technical stuff for people who like algorithms

Endgame analysis

Monotonicity

Subadditivity in easy mode

Timing examples (also illustrates the different definitions of "solve")

Track of wordlist changes

The Best Strategies for Wordle, part 2

This follows on from the January 2022 piece [The best strategies for Wordle](#).

There is a follow-up article here: [The best strategies for Wordle, part 3 \(July 2023\)](#).

Note that as part of Wordle being taken on by the New York Times on 15 February 2022, the wordlists were modified to remove a few offensive and obscure words. Another change was made on 16 March 2022, partly reverting the changes of 15 February and reordering the hidden wordlist (see below). These changes don't affect the best starting word, **SALET**, and only make minimal difference to the best strategy as a whole. Unless otherwise stated, all results in this post relate to these New York Times wordlists as of 16 March 2022. (Judging by the pattern of the changes to the wordlists, I wouldn't be surprised to see further changes in the future.)

Update 10 September 2022: On 30 August 2022, the New York Times expanded the guessing list by adding 1881 brand new obscure words, because the existing guessing list obviously didn't get obscure enough. (The hidden word list remains unchanged at 2309 entries.) Now you can enrich your life by guessing with words like **PTISH**, **OUREY** and **PEKAU**. Not only that, you can also use **TARSE**

which turns out to be slightly better than **SALET** as a first guess, thus becoming the new number one starting word in easy mode. (**SALET** retains the top spot in hard mode.) See the [section below](#) for further details.

I received some feedback from the [January piece](#). Here are some points that arise from that, along with some other comments:

- This isn't Wikipedia - it's just my blog. It has a similar look to Wikipedia because I'm using the same publishing mechanism, called [Mediawiki](https://www.mediawiki.org/wiki/MediaWiki) (<https://www.mediawiki.org/wiki/MediaWiki>).
- There are a lot of claims flying around the internet involving the words "optimal" and "best play" which are almost all wrong (or, being generous, a little misleading). Typically someone will optimise their favourite heuristic one move ahead (e.g., choose the word that minimises the the number of possible words at the next stage, given the least favourable colour score) and call it job done - the words **SOARE** and **ROATE** often feature in this context. These are not bad words (being equal 54th best and equal 61st best respectively), but they aren't the best in a commonly-agreed sense. The measure used in this post is to minimise the average number of guesses required when up against a randomly chosen hidden word from the allowable list. This involves analysing all possible games right to the end, and proving the result is correct. While this isn't the only possible measure, it seems like a reasonably natural/objective one - a idea of how to play the game in the best way that many people could agree with.
- Of course people mostly don't care about playing optimally - they play to have fun. All the same, I think there is an element of curiosity about the best way to play in any situation. Are you really stuck, or is there a way out? Were you unlucky or was there something you could have done earlier to prevent the problem? How can you get one over your relative/colleague/friend/enemy/...? Since there is a proliferation of articles about the best word "from a human point of view", perhaps there are some general principles that may be gleaned from computer analysis that apply to human-style play, without taking any fun away. If you are interested in this then you may like to visit the [section below](#).
- From my point of view, the fun is in designing an algorithm to manage the apparently large search space. If there is an estimate that it would take a CPU-year to run through all the possibilities, then it's very likely that it can be done in a CPU-day, because a few orders of magnitude algorithmic improvement are always there for the taking if you want them. And after that, a few more orders of magnitude are going to be available too. This is a general principle. Algorithms are floating around everywhere - you just have to reach out and take them.
- For convenience, I'm referring to the default (non-hard) mode, where any word is allowed as a guess, as "easy mode". This isn't meant to imply it's actually easy, but it's a lot more concise than saying "the default (non-hard) mode, where any word is allowed as a guess" every time.
- I've made a technical section below for those interested in methods, timings and so on. Read on as far as you want to.
- The colour scores will be referred to as B, G and Y, for Black, Green and Yellow, with Black representing a wrong letter. I'm using this convention because that's how it was the first time I saw it, though other renditions have the colour score of a wrong letter as white or grey. If it helps, you can think of B as "blank".

Updated best starting words using New York Times word lists as of 30 August 2022

See below for complete lists of values of all possible starting words.

Top 10 first words for Wordle in easy mode, word lists as of 30 August 2022

Rank	First word	Average guesses required	Total guesses required over all possible hidden words
1	TARSE	3.4140	7883
2	SALET	3.4188	7894
3	REAST	3.4197	7896
=4	CRATE	3.4210	7899
=4	SLATE	3.4210	7899
=4	TRACE	3.4210	7899
7	CRANE	3.4223	7902
8	CARLE	3.4253	7909
=9	CARNE	3.4288	7917
=9	SLANE	3.4288	7917

Top 11 first words for Wordle in hard mode, word lists as of 30 August 2022

Rank	First word	Average guesses required	Total guesses required over all possible hidden words
1	SALET	3.5067	8097
2	LEAST	3.5097	8104
3	REAST	3.5132	8112
4	CRATE	3.5162	8119
5	TRAPE	3.5171	8121
6	TARSE	3.5180	8123
7	SLANE	3.5188	8125
8	PRATE	3.5206	8129
9	CRANE	3.5214	8131
=10	CARLE	3.5236	8136
=10	TRAIN	3.5236	8136

These are evaluations from an exhaustive search - i.e., proved to be optimal for the average number of guesses. The second column is the third column divided by 2309, the number of possible hidden words.

Complete lists showing the average number of guesses required for the optimal strategy for all 14855 starting moves are available here: [Easy mode results](https://github.com/alex1770/wordle/blob/main/results_easy_nyt20220830) (https://github.com/alex1770/wordle/blob/main/results_easy_nyt20220830) and [Hard mode results](https://github.com/alex1770/wordle/blob/main/results_hard_nyt20220830) (https://github.com/alex1770/wordle/blob/main/results_hard_nyt20220830).

These are the corresponding strategy files for **TARSE** in easy mode (<https://raw.githubusercontent.com/alex1770/wordle/main/tarse.easy.nyt20220830.tree>) and **SALET** in hard mode (<https://raw.githubusercontent.com/alex1770/wordle/main/salet.hard.nyt20220830.tree>). See [the previous piece](#) for how to use these files to play optimally.

Updated best starting words using New York Times word lists as of 16 March 2022

See below for complete lists of values of all possible starting words.

Top 11 first words for Wordle in easy mode, using NYT word lists

Rank	First word	Average guesses required	Total guesses required over all possible hidden words
1	SALET	3.4201	7897
2	REAST	3.4214	7900
3	CRATE	3.4223	7902
4	TRACE	3.4227	7903
5	SLATE	3.4236	7905
6	CRANE	3.4244	7907
7	CARLE	3.4275	7914
8	SLANE	3.4301	7920
=9	CARTE	3.4327	7926
=9	SLANT	3.4327	7926
=9	TORSE	3.4327	7926

Top 10 first words for Wordle in hard mode, using NYT word lists

Rank	First word	Average guesses required	Total guesses required over all possible hidden words
1	SALET	3.5076	8099
2	LEAST	3.5106	8106
3	REAST	3.5132	8112
4	CRATE	3.5171	8121
5	TRAPE	3.5180	8123
6	SLANE	3.5193	8126
7	PRATE	3.5210	8130
8	CRANE	3.5223	8133
=9	CARLE	3.5236	8136
=9	TRAIN	3.5236	8136

These are evaluations from an exhaustive search - i.e., proved to be optimal for the average number of guesses. The second column is the third column divided by 2309, the number of possible hidden words.

Exhaustive evaluations showing the average number of guesses required for the optimal strategy for all 12972 starting moves is available here: [Easy mode results](https://github.com/alex1770/wordle/blob/main/results_easy_nyt20220316) (https://github.com/alex1770/wordle/blob/main/results_easy_nyt20220316) and [Hard mode results](https://github.com/alex1770/wordle/blob/main/results_hard_nyt20220316) (https://github.com/alex1770/wordle/blob/main/results_hard_nyt20220316).

These are the corresponding strategy files for **SALET**: [easy mode](https://raw.githubusercontent.com/alex1770/wordle/main/salet.easy.nyt20220316.tree) (<https://raw.githubusercontent.com/alex1770/wordle/main/salet.easy.nyt20220316.tree>) and [hard mode](https://raw.githubusercontent.com/alex1770/wordle/main/salet.hard.nyt20220316.tree) (<https://raw.githubusercontent.com/alex1770/wordle/main/salet.hard.nyt20220316.tree>). See [the previous piece](#) for how to use these files to play optimally.

What makes a good strategy from a human point of view?

What can we glean about the kinds of human-friendly strategies that are likely to work well? Here we're relaxing the idea of sticking rigidly to the perfect/optimal strategy, instead sifting through the computer output to try to learn some general principles or rules of thumb, so arriving at a compromise between a generally sound strategy while still allowing for a decent amount of personal preference in style of play to keep it fun.

- Remember that the target word will be an ordinary (non-obscure) five-letter word that is not derived in a simple way from a smaller word: there are no plurals ending in 's' or past tenses that are formed just by adding 'ed'.
- The first word should contain different letters from E, A, T, R, S, L, N, C, I, O with earlier letters in this list generally preferred to later ones. (U is not as good as you might think.)

- The first word should ideally contain two vowels from A, E, I, O. Using one or three vowels may be OK, but four is definitely poor according to the computer. (Popular choices like **AUDIO** or **ADIEU** aren't actually theoretically very sound, though of course they may still suit your style if you prefer to nail down the vowels early.)
- You can relax about trying to choose the right answer on the second guess as it requires a large dose of luck to get it in two. You are usually still discovering letters on this guess, so you can choose a completely new set of five letters, or you can include some Yellow/Green-scoring letters from the first go (obviously try a new location for a Yellow-scoring letter to try to narrow down its location). It turns out that either of these two methods, a new set of letters or a compatible word, lead to approximately similar performance in terms of the average number of guesses required, provided no letter is wasted. (You don't want to choose an already-eliminated letter, or choose a rare letter like Z at this stage. Of course if you didn't discover any vowels in the first guess, it makes sense to add some new ones in the second guess.)
- By guess three there should only be a handful of words that fit, because even if you got mostly blanks for the first two guesses then at least you have eliminated common letters. It's a good idea, if you can, to make yourself write down a few possible hidden words, and then pick the one that, should it be wrong, is most informative about the others.
- Occasionally, though, even with a lot of letters discovered it may be necessary to take "insurance": expending one guess that you know can't be the hidden word in order to distinguish between a lot of similar words in one go. For example, recently the hidden word was **SHAKE**. If you start off by guessing **STAGE** and receive **GBGBG**, and next guess **SHAPE** and get back **GGGBG**, then the possible hidden words are **SHADE**, **SHAKE**, **SHALE**, **SHAME**, **SHARE** and **SHAVE**. You obviously don't want to be guessing these one-by-one as you might end up needing eight guesses, so in this situation you should enter a guess that you know can't be the hidden answer. A word like **LOVED** works well because it contains three of the letters that can occur in position four.
 - (Aside for hard mode.) In hard mode it wouldn't be legal to guess **LOVED** here, and you would be trapped in a bad endgame. Really the problem lies earlier: at guess two you would need to avoid an **H** in the second position which might get fixed by a **G** reply, though planning this far ahead is possibly getting a bit esoteric. The best second guess here (after **STAGE** receives **GBGBG**) is **SPARE** with **P** and **R** covering two of the possibilities for **SHA?E**. If you continue to get replies of **GBGBG**, then successively **SLAKE**, **SUAVE** and **SHADE** guarantee solving it in six guesses as can be seen from this strategy file (<https://github.com/alex1770/wordle/blob/main/stage.gbgbg.tree>).

Statistics that inform some of the above rules of thumb:

Frequency of letters in the hidden word list

Rank	Letter	Frequency	Total number of occurrences in 2309-long word list
1	E	0.107	1230
2	A	0.084	975
3	R	0.078	897
4	O	0.065	753
5	T	0.063	729
6	L	0.062	716
7	I	0.058	670
8	S	0.058	668
9	N	0.050	573
10	C	0.041	475
11	U	0.040	466
12	Y	0.037	424
13	D	0.034	393
14	H	0.034	387
15	P	0.032	365

Frequency of letters in top 101 starting words

Rank	Letter	Frequency	Total number of occurrences in top 101 starting words
1	E	0.158	80
2	A	0.149	75
=3	T	0.139	70
=3	R	0.139	70
5	S	0.101	51
6	L	0.075	38
7	N	0.063	32
8	C	0.055	28
9	I	0.048	24
10	O	0.040	20
11	P	0.018	9
12	D	0.010	5
13	U	0.004	2
14	H	0.002	1

(Reference: the [full list of best starting words in easy mode](https://github.com/alex1770/wordle/blob/main/results_easy_nyt20220316) (https://github.com/alex1770/wordle/blob/main/results_easy_nyt20220316)).)

It looks like the best letters to use for a starting word don't correspond exactly to the more frequent letters in English five letter words (or what is similar, the frequency of letters in the 2309 long list of possible hidden words). E, A, R and T are at or near the top of both lists, but the vowels O and I don't feature prominently amongst the best starting words, and U is virtually non-existent. Of the top 101 starting words, 80 have two vowels (11 have one, and 10 have three). S is popular amongst the best words, perhaps surprisingly so, given that the hidden word list has no simple plurals.

Analysis of your play

If you want to tighten up your play, or you're just curious as to how lucky or unlucky you were, you can use the program (<https://github.com/alex1770/wordle>) in analysis mode. This separates out how well you did into skill and luck components. For example, if after some number of guesses the hidden word can be found with an average of 1.3 more guesses with best play, but you choose a word that would require an average of 1.5 more guesses, then that would count as a slight inaccuracy of 0.2 guesses, regardless of whether or not that particular choice happened to turn out well subsequently.

For example, typing `wordle -A crate.ybbbb.lions.bbybg.focus.ggggg` (or `wordle -A crate.lions.focus` for short) would produce:

crate: Near perfect choice	(Inaccuracy = 0.0022 guesses)	Best choice was salet
ybbbb: Average luck	(Luck = 0.0223 guesses)	
lions: Good choice	(Inaccuracy = 0.0800 guesses)	Best choice was spoil
bbybg: Lucky	(Luck = 0.4800 guesses)	
focus: Perfect choice!	(Inaccuracy = 0.0000 guesses)	Best choice was focus
ggggg: Average luck	(Luck = 0.0000 guesses)	

Total rating of word choices: Near perfect choices (Total inaccuracy = 0.0822 guesses)
 Total luck from colour scores: Lucky (Total luck = 0.5023 guesses)

which corresponds to a situation where you chose **CRATE** and received YBBBB, then chose **LIONS** and so on. It is telling you that your word choices were pretty good, though you got a bit lucky with the BBYBG score.

Or another example, wordle -A salet.bybbb.handy.bygbb produces:

salet: Perfect choice!	(Inaccuracy = 0.0000 guesses)	Best choice was salet
bybbb: Slightly unlucky	(Luck = -0.1344 guesses)	
handy: Bad choice	(Inaccuracy = 0.4257 guesses)	Best choice was brond
bygbb: Lucky	(Luck = 0.9802 guesses)	

Total rating of word choices: Not great choices (Total inaccuracy = 0.4257 guesses)
 Total luck from colour scores: Lucky (Total luck = 0.8458 guesses)

highlighting the poor choice **HANDY**, which neglected to vary the position of the A from the first guess.

Choosing two words at a time

Several people have asked, what are the best two starting words if the second word doesn't depend on the score (the five colour response) from the first word? I don't imagine many people will be dedicated enough to memorise all the best second words to meet each possible colour-response to the first word **SALET**, but if you wanted a decent start that doesn't involve any effort then just using the best pair of words isn't too bad. It turns out that the best pair is **PARSE CLINT**, which if best moves are played subsequently would require an average of 3.5955 guesses using the NYT (2022-03-16) word lists. This compares with the 3.4201 from **SALET**, which means that ignoring the first score costs you about 0.18 guesses on average. (Since you asked, [clints](https://en.wikipedia.org/wiki/Limestone_pavement#Formation_of_a_limestone_pavement) (https://en.wikipedia.org/wiki/Limestone_pavement#Formation_of_a_limestone_pavement) are "a series of limestone blocks or ridges divided by fissures or grikes" - definition from [Chambers Dictionary](https://chambers.co.uk/) (<https://chambers.co.uk/>)).

It doesn't take all that long to evaluate all pairs, with a suitable value threshold. My fairly ordinary desktop computer only required about an hour to run through all 84 million pairs of words and find the top 11,099 (I was going for a round 10,000, but overshot slightly) with proof of their evaluation, and with proof of completeness of the list. For anyone interested, this is the list of the top 11099 pairs (<https://github.com/alex1770/wordle/blob/e298e35fbc044a8137da235a8940748770cefd2c/twowords-nyt20220316-best11099.txt>).

Top 10 two-word guesses for Wordle in easy mode, using NYT word lists

Rank	First word	Average guesses required	Total guesses required over all possible hidden words
1	PARSE CLINT	3.5955	8302
2	CRANE SPILT	3.6003	8313
3	CRINE SPALT	3.6016	8316
=4	SLANT PRICE	3.6037	8321
=4	LARNT SPICE	3.6037	8321
=6	SLANT CRIPE	3.6046	8323
=6	CRANE SLIPT	3.6046	8323
8	CRISE PLANT	3.6063	8327
=9	CRINE PLAST	3.6068	8328
=9	PRASE CLINT	3.6068	8328

(It took me too long to realise that the top 10 word pairs are all anagrams of **PARSE CLINT**.)



Clints

© M J Richardson (<https://www.geograph.org.uk/photo/2865373>)

CC BY-SA 2.0 (<https://creativecommons.org/licenses/by-sa/2.0/>)

Using larger hidden word lists

Some people regard it as slightly unfair for a program to make use of the fact that the hidden word has to be on the 2309-long hidden word list, and they prefer to look for strategies that would work with a wider possible set of hidden words. Laurent Poirrier has written a nice account (<https://www.poirrier.ca/notes/wordle-optimal/>) of the various possibilities, in particular discussing the version where any possible word that can be used as a guess word is considered as a possible hidden word. I call this "bighidden" (not the greatest choice of names, but it's a way of talking about it). The answers described here (<https://puzzling.stackexchange.com/questions/114316/whats-the-optimal-strategy-for-wordle/114686>) by Mark Fisher and others also mostly consider this bighidden version of optimal wordle. A summary of my results can be found on that same StackExchange post, here (<https://puzzling.stackexchange.com/questions/114316/whats-the-optimal-strategy-for-wordle/115599#115599>).

As to whether this (using the full allowable guess list as the hidden word list) is the "true" problem or not, I think it's a reasonable thing to analyse because it's good to be robust and not to overfit a method to particular data set. On the other hand, the argument I'd make in favour of restricting to the 2309-long hidden word list of words that can appear, which is the situation discussed in the other sections of this piece, is that I think this list is a fairly good match for the subset of ordinary words that people tend to know, so corresponds most closely to people's expectations and how they play in practice. The game creator, Josh Wardle, did a good job of segregating a large list of what you might call "Scrabble words" - around 13000 mostly obscure words - by extracting around 2300 ordinary words. To illustrate this, here are 20 random words from the 2309-long list: *plied wrung flesh grave scout*

quake musky lying least raven hutch canny taker hardy batty croak maybe squad blunt chase, which for my money are all fairly normal words. Here, on the other hand, are 20 random words from the remaining 10638-long list: *alcid tallas ashet runch tally emics namus seely chico noils winge doges oread ruche faffy telia lucks sampi arepa myops*, which apart from the plurals and maybe one or two others, are not what people are expecting to find getting the five greens of the day.

But it's still a good challenge to try using the larger list of 12972 words as the possible hidden set. It makes analysis considerably harder, and answering one of these questions (the precise hard mode result) required upgrading the code from the original to be around 100x more efficient by using on-the-fly proving methods where the program effectively is searching through proofs as well as just enumerating possibilities.

Note that since the New York Times reverted the guessing wordlist on 16 March 2022 to the original list of 12972, and since this section only concerns the guessing wordlist, not the hidden wordlist, the results in this section apply to the "classic Wordle" (pre 15 February 2022) wordlist as well as the 16 March 2022 wordlists.

I find these results using "bighidden" (12972 guess words, 12972 possible hidden words):

- It's possible to solve (easy mode) Wordle in 6 guesses, but not in 5, confirming Laurent's and Alex Peattie's results. (My program will find 6-guess answers at the rate of about one every minute, which compares favourably with 1000 hours or so required by Laurent's method, though no doubt Laurent could improve the efficiency if he wanted to. I think this is an interesting illustration of the kind of improvements available and I have no doubt that it could be made a lot more efficient still if that were a desirable goal.) Extending Laurent's results, there are at least 2109 starting words that guarantee 6 guesses (previously, only two or three were known?), listed here (<https://github.com/alex1770/wordle/blob/main/out-nyt20220316-easy-bighidden-g6-depthonly-n250-goodlist>). These were obtained using a heuristic (non-exhaustive search) trying each of the 12972 possible starting words, so it's possible that there may be a few more (likely not many more) than 2109 that work, but all the starting words in this list of 2109 are known to use 6 guesses.
- In easy mode with 6 guesses, the starting word **LANTS** requires an average of 4.07771 guesses (total of 52896 over all hidden words), and this is proven to be exact, i.e., optimal for **LANTS**, or in other words, the result of an exhaustive search starting with **LANTS**. I think this is likely to be the best starting word, minimising the average number of guesses required, but I haven't done an exhaustive search starting with every other word so there could conceivably be a better one. **LANTS** is probably the best because it comes out far enough ahead of other words on an incomplete search that is usually very accurate, but it hasn't been proved to be the best. A strategy file for **LANTS** is available here (<https://github.com/alex1770/wordle/blob/main/lants.bighidden.origwords.tree>). (Explanation of how to use these strategy files can be found in the previous article.)
- In hard mode you can do it in 7 guesses, and this is optimal (6 is impossible) which settles the question of the exact number of guesses required. (Previously the best known (<https://www.poirier.ca/notes/wordle/#variant-hard-mode>) was ≤ 14 guesses?) There are only seven starting words that allow you to solve it in 7 guesses: **BLAWN**, **CHOMP**, **GUMBO**, **THUMB**, **THUMP**, **TUPIK**, **WHOMP** - the remaining 12965 have been proven to fail.
- In fact, in hard mode with a limit of 7 guesses, the best starting word in terms of average number of guesses is **THUMP**, which requires an average of 4.52629 guesses (total of 58715, proved correct and better than any other starting word), so hard mode can be considered completely solved from this point of view. A strategy file can be found here (<https://github.com/alex1770/wordl>

The hard mode result came as a bit of a surprise as I thought it was going to end up requiring 8 guesses. The original searcher program (`wordle.cpp`) from January proved fairly quickly that 6 guesses were impossible, and found some examples of words that solved it in 8 guesses, also fairly quickly (minutes). So the only question was whether the minimum number of guesses required was 7 or 8. I then ran a depth 7 search with fairly wide heuristic cutoffs to try to find a starting word that would require only 7 guesses, but it failed to find anything. Since this kind of search is usually fairly accurate, I guessed that the correct answer was going to be 8, and the way to prove this would be a complete exhaustive search at depth 7. But as this was projected to take several months on my equipment (a single desktop computer), the algorithm needed to be improved by some more orders of magnitude first(*). Having done that, running the depth 7 exhaust yielded a surprise: within a day or two it found 7 starting words that work within 7 guesses and demonstrated that these were the only ones. These are a little tricky to find, since not only are the starting words themselves few and far between, the follow-up words are sometimes very rare and counterintuitive too. For example, if **CHOMP** receives a score of **BBBBB**, there are only two words out of 12972 that ensure success within 7 guesses: **BEZIL** and **KUDZU!** Common sense would say that using a rare letter like Z, or repeating a letter that is not known to be present (U), should be bad moves, but perhaps in hard mode with not many available guesses there are different priorities and it's more important to avoid getting trapped into a bad endgame.

(*) I tried to limit run times to a day or two. Anything longer and my daughter will start to complain that her Minecraft computer (formerly my computer) is laggy too much of the time. I'd also prefer to stick to an ordinary desktop rather than use a compute farm of some sort, since the interest is more in the algorithm than the result.

Technical stuff for people who like algorithms

The program used is available here (<https://github.com/alex1770/wordle/blob/main/wordle.cpp>). The basic overview is that there are two functions that recursively call each other (simplified for clarity):

```
minoverwords(list oktestwords, list hiddensubset, int remainingguesses, int beta){...}  
sumoverpartitions(list oktestwords, list hiddensubset, int remainingguesses, word testword, int beta){...}
```

The entry point (if there are 6 allowable guesses) is: `minoverwords(T, H, 6, infinity)` where usually T will be the set of 12972 allowable guessing words, and H will be the set of 2309 allowable hidden words. (In "bighidden" mode, H will be the full set of 12972 guessing words.) `minoverwords(T, H, g)`, called $m(T, H, g)$ for short, evaluates the minimum over all strategies requiring no more than g guesses of the sum over h in H of the number of guesses required for h, with infinity meaning that there is no way to guarantee solving it within g guesses.

Python-like pseudocode:

```

def minoverwords(T,H,g,β=infinity):
    if g==0: return infinity
    for w in T:
        β=sumoverpartitions(T,H,g-1,w,β)
    return β

def sumoverpartitions(T,H,g,w,β):
    Let s_1,...,s_k denote the possible colour scores (GBYBG etc) of H with the trial word w,
    and H_1,...,H_k denote the corresponding subsets of H
    t=0
    for i in {1,...,k}:
        t=t+minoverwords(filter(T,w,s_i), H_i, g, β)
        if t>=β: return β
    return t

def filter(T,w,s):
    If easy mode: return T
    If hard mode: return {t in T | t is an allowable test word in hard mode given that word w has scored s}

```

The purpose of beta (β) is an optimisation, similar to [alpha-beta pruning](https://en.wikipedia.org/wiki/Alpha-beta_pruning) (https://en.wikipedia.org/wiki/Alpha-beta_pruning) in an adversarial game, the idea being that once you've found a refutation of a move you don't need to carry on searching for the optimal refutation. Here in wordle, once you know you have a refutation of a word choice, you don't need to carry on searching to quantify the exact extent of the refutation. In wordle we're dealing with min-sum rather than min-max which means (1) it's harder to get an analogue of alpha, and (2) you have to do extra work to make the beta cutoff ($t \geq \beta$ condition) happen usefully often (see below).

To expand on (1): with min-max you'd be able to reason that there is no need to evaluate x in expressions like $\max(5, \min(4, x))$, but here we have expressions like $a + \min(b, x)$ which will depend on x for any finite values of a and b . (If we are only interested in whether wordle can be solved within a certain number of guesses, then effectively the only values are 0 and infinity, and sum becomes max, which means we can do alpha cutoffs, exiting the min loop early.)

The above is the very basic algorithm, which will run many orders of magnitude too slowly to get any results mentioned in the above sections. Fortunately, there are many easy ways to improve this, some of which are listed below:

- To get the cutoffs, $t \geq \beta$, to occur usefully often in `sumoverpartitions()`, it's much better first to get fast lower bounds for all the summands. That is, we can make several passes over the w loop, first picking off some fast lower bounds, then some not-quite-so-fast lower bounds, then evaluating `minoverwords(filter(T,w,s_i), H_i, g, β)` in full. If at any point the sum of the current lower bounds and exactly-calculated values exceeds β , then we can return immediately.
- A basic lower bound is $m(T, H, g) \geq 2|H| - 1$ because this is the total number of guesses required if your test word is in H and if this word makes the next choice unique (splits the rest of H into singletons). More careful lower bounds can be found by considering under what conditions a total of $2|H|$ and $2|H| + 1$ can be obtained.
- It's a big gain to cache the values $m(T, H, g)$, because the same T , H and g can arise in many different ways. For example, if **SALET** gets the score **YBBBB**, it's exactly the same as **SLATE** getting the same score **YBBBB**. In both cases we know there is an **S**, not in the first position, and no **A**, **L**, **E** or **T**. The overall value, $m(T, H, g)$, as returned by `minoverwords()` with `beta=infinity`, is a function of T , H and g , but it's only really function of T in hard mode. In easy mode, T is a constant throughout the run so can be ignored. The T -dependence in hard

mode makes the cache less useful, and more profligate on memory, but even so, it's still very useful.

- It helps to order the words w in T in `minoverwords()` in order of the best (as estimated by some heuristic) first. This means β drops low early on in the loop, which speeds up the other calls. It also leads to earlier exits from that loop in situations where we know lower bounds for $m(T, H, g)$. A case in point is when we only care about finding the hidden word within a certain number of guesses, rather than the average number of guesses required, in which case we can exit the loop as soon we find a word that works. Another case is when we can calculate lower bounds on the total number: $2|H|-1$ as mentioned above is the basic one, but this can be improved.
- Similarly it helps to order the i loop in `sumoverpartitions()`, though in this case it turns out that it is much better to order by increasing $|H_i|$ even though larger $|H_i|$ are more likely to lead to larger increases in the current lower bound for `minoverwords(filter(T, w, s_i), H_i, g, β)` and cause a cutoff. The point is that lower H_i are much faster to evaluate, and you tend to get a bigger increase in t per unit time by evaluating these first.
- As well as storing exact values for $m(T, H, g)$ in the cache, there are also opportunities for storing lower bounds. If β is never reduced in the `w` loop in `minoverwords()` then all the calls to `sumoverpartitions()` must have returned values that were clipped at β , i.e., representing $\geq \beta$, which means the final returned value of β is a lower bound for $m(T, H, g)$. Lower bounds are then used in the i loop in `sumoverpartitions()` to make the cutoff $t \geq \beta$ happen earlier. There are other opportunities for calculating lower bounds, described below.
- Endgame analysis (see below)
- Monotonicity (see below)
- Subadditivity (see below)

The last three of the above methods are a bit more elaborate and are explained in more detail below.

Endgame analysis

As mentioned in the section above, the solver can sometimes run into a problem when the current hidden word list (H) contains an awkward subset where four of the letters are fixed. This mainly applies in "bighidden" mode, where the full set of test words is used as the hidden word set (i.e., initially $H=T$).

In the example above, there were seven words fitting the pattern **SHA?E**, of which only one had been eliminated at the moment in question, and this forced the solver to choose words that matched more than one of the possible **?** letters at once, but in bighidden mode, the pattern **?ILLS** matches nineteen different words (remembering that T contains what we might call "Scrabble words", so includes some obscure ones), and there are many patterns, such as **CO?ED** and **?INES**, that match twelve or more words. We'll call a pattern like **?ILLS**, **CO?ED** or **?IGHT** an *endgame*. When the number of remaining guesses is limited, the existence of such endgames as subsets of your current set of possible hidden words, H , can significantly restrict your choice of guess word, because you have to be able to distinguish between the words in every endgame. The only way to distinguish between (e.g.,) **EIGHT**, **FIGHT**, **LIGHT**, **MIGHT**, **NIGHT** and **TIGHT** is to use guess words that include at least five of the six letters **E**, **F**, **L**, **M**, **N** and **T**. (This is a necessary but not sufficient condition, because a **T** in a guess word may not tell you anything about an initial **T** in the hidden word, if there is only one **T** in the guess word and if it's not in the first position.) This reasoning is the basis of Alex Peattie's proof (<http://www.alexp.org/~alexpeattie/scrabble/>)

[s://alexpeattie.com/blog/establishing-minimum-guesses-wordle/](https://alexpeattie.com/blog/establishing-minimum-guesses-wordle/)) that five guess words is not sufficient, making use of the endgame **?ILLS**. The task here is to get it to use all endgames, and to automate it to work in real time as part of the solver, so it speeds up the search in other situations with more than five available guesses.

Imagine being in the middle of a run that is trying to evaluate the average number of guesses required, or whether the initial conditions can be solved within some number of guesses. Let's say the program flow is somewhere at the start of `minoverwords()`, and E is some endgame (e.g., $E = \{\text{EIGHT}, \text{FIGHT}, \text{LIGHT}, \dots\}$). Then we look for the endgame with the largest intersection $L = E \cap H$. Having chosen E (depending on L), we can then count the number of distinct colour scores $n(L, w) = |\{score(e, w) | e \in E\}|$ for each word w in T , where $score(e, w)$ is the colour score (YBBBG etc) of hidden word e with guess word w . If there are currently g guesses available, then by the time we've used $g-1$ guesses, we need to have reduced the set of possibilities from L to (at most) a single choice. Since w cuts down the choices by at most $n(L, w) - 1$ starting from any subset of L , we know that the current configuration is not solvable if the sum of the $g-1$ largest values of $n(L, w) - 1$ (as w varies over T) comes to less than $|L| - 1$.

This is a sufficient condition for a "static" early exit, but a more refined analysis might allow it to exit early more often, if the guess words w don't work efficiently together. To make use of this possibility, if there was no static exit, the program can decide to do a complete run using L in place of H : if this is insoluble within the prescribed number of guesses, then the original H must also be insoluble. Whether or not it attempts this trial run is decided on the basis of a simple heuristic that gauges the likely benefit vs the time spent.

Monotonicity

The more guessing words you have, and the fewer possible hidden words there are, the better off you are. In other words, if $T' \supset T$ and $H' \subset H$, then $m(T', H', g) \leq m(T, H, g)$. What's that good for? It's not particularly easy to do a cache lookup based on supersets/subsets like this, so how can this inequality usefully be applied? One useful case is this: "It's better to introduce a new letter, even if it scores B, than to waste a letter by (re)using a letter known not to be in the hidden word."

For example, in hard mode, if the state **NORTH.BBBYB.TUNIC.YBBBB** (meaning **NORTH** receives a score of **BBBYB** then **TUNIC** receives a score of **YBBBB**) is insoluble within the prescribed number of guesses (has value infinity), then **NORTH.BBBYB.TONIC.YBBBB** will also be insoluble without any further calculation. The set of permissible guessing words is the same in both cases, being any word with a **T** in it (due to the way hard mode rules work), and the set of possible hidden words for **NORTH.BBBYB.TUNIC.YBBBB** is a subset (namely those words without a **U**) of those for **NORTH.BBBYB.TONIC.YBBBB**. This kind of rule is actually very easy to apply and also very useful. A great deal of time in a search is spent trying each of the 12972 words in the min loop in `minoverwords()`, trying to improve on the best word so far, even though with any decent heuristic ordering an improvement after the first few hundred words is unlikely to happen. But to ensure a watertight final answer, every one of these words has to be tried, no matter how apparently poor it is. A proof-type method like the one described here means that many bad choices can be discarded without any further search. This makes a big difference in some of the longer searches in bighidden mode.

Subadditivity in easy mode

In easy mode, where the set of guess words, T , doesn't change, we can say more. If H_1, \dots, H_k partition H , then $m(T, H_1, g) + \dots + m(T, H_k, g) \leq m(T, H, g)$. (Actually this is true in hard mode too, but unlikely to be useful because you would normally have to use different T sets, so the expression on the left won't arise in practice.)

The meaning of this inequality is that it can't hurt to know something extra about the hidden word, because you can always choose to ignore it. (A proof is that the strategy file, or proof certificate, for H can be applied separately to each H_i .)

For example, if we understand **NORTH.BBBYB.TUNIC.YBBBB** to represent the state of (T, H, g) after those two guesses and colour scores, then monotonicity gave us $m(\text{NORTH.BBBYB.TUNIC.YBBBB}) \leq m(\text{NORTH.BBBYB.TONIC.YBBBB})$, but subadditivity tells us more:

$$m(\text{NORTH.BBBYB.TUNIC.YBBBB}) + m(\text{NORTH.BBBYB.TUNIC.YYBBBB}) + m(\text{NORTH.BBBYB.TUNIC.YGBBB}) \leq \\ m(\text{NORTH.BBBYB.TUNIC.Y*BBB}) = m(\text{NORTH.BBBYB.TONIC.YBBBB}).$$

In the above, '*' is a wildcard, meaning that the state **NORTH.BBBYB.TUNIC.Y*BBB** refers to the set of hidden words, H , where the colour score for the U can be anything. The states **NORTH.BBBYB.TUNIC.Y*BBB** and **NORTH.BBBYB.TONIC.YBBBB** are the same as each other, because there is no new information from the U of TUNIC (because of the wildcard) or the O of TONIC (because the letter O is already known not to be present from **NORTH.BBBYB**). Note that the inequality doesn't work in hard mode, because the allowable guesses, T , will be more restricted for components of the left hand side, which could make them worse off.

New lower bounds that arise in this way can be stored in the lower bound cache, and used later to exit the sum-loop in `sumoverpartitions()` early. These bounds are useful when you are optimising the average (or total) number of guesses. This is in contrast to the monotonicity bound, above, which is more-or-less only useful in "depth only" mode: where the only question is whether there is a solution that is guaranteed to work within a certain number of guesses.

Timing examples (also illustrates the different definitions of "solve")

Here are four different versions of what it means to "find" the best starting word, together with the corresponding command line instruction (using the [program here \(<https://github.com/alex1770/worlde>\)](https://github.com/alex1770/worlde)), and the CPU time taken. The fifth entry represents finding the value (average number of guesses used) for all 12972 starting words. In all cases, the NYT (2022-03-16) word lists are used.

Times are all *single core* CPU times to make the timings more comparable across different computers. Using multiple processes for the long runs (with 12972 starting words) is easy to do (just assign different starting words to different cores). Long runs parallelise tolerably well (though not perfectly)

as each process will fairly quickly build up its own useful cache of values that wouldn't be enormously improved by combining with the caches of the other processes. (An improvement would be to multithread it and share the cache efficiently.)

The computer used is an Intel i7-4930K @ 3.40 GHz, with a P9X79 motherboard and 48 GiB DDR3 1866 MHz RAM. This configuration is getting on a bit (circa 2013) - more modern setups may be a little faster.

Command	Time	Description
wordle -n2	0.83 seconds	Find the best starting word, SALET , without proof that it is the best word, and without finding its optimal average number of guesses
wordle -w salet -n8 -p salet.tree	0.37 seconds	Find the optimal number of guesses for SALET (3.4201 on average) together with its decision tree, without proof of its optimality
wordle -w salet	1.19 seconds	Prove 3.4201 guesses is optimal for SALET
wordle -c10	15 hours	Prove SALET is the optimal word out of 12972 (prove all other words result in at least 3.4201 guesses)
wordle -c10 -s	6 days	Find, with proof, the optimal average number of guesses for all 12972 starting words

Track of wordlist changes

Dates	List of guessing words	List of hidden words
Pre 15 Feb 2022	12972 (https://github.com/alex1770/wordle/blob/main/wordlist_nyt20220316_all)	2315 (https://github.com/alex1770/wordle/blob/main/wordlist_hidden)
15 Feb - 15 March 2022	12947 (https://github.com/alex1770/wordle/blob/main/wordlist_nyt20220215_all)	2309 (https://github.com/alex1770/wordle/blob/main/wordlist_nyt20220316_hidden)
16 March - 29 August 2022	12972 (https://github.com/alex1770/wordle/blob/main/wordlist_nyt20220316_all)	2309 (https://github.com/alex1770/wordle/blob/main/wordlist_nyt20220316_hidden)
Post 30 August 2022	14885 (https://github.com/alex1770/wordle/blob/main/wordlist_nyt20220830_all)	2309 (https://github.com/alex1770/wordle/blob/main/wordlist_nyt20220830_hidden)

Notes:

- Note that the 12972-long list of guessing words was reverted on 2022-03-16 to what it was originally. (There is only one list of each length - 2309, 2315, 12947, 12972 - so the length can be used as a name for the list.)
- The lists above have been sorted into alphabetical order, because the analysis here doesn't depend on the order. (As it happens, the post 16 March list of hidden words used by the actual game was re-ordered.)

Retrieved from "http://sonorouschocolate.com/notes/index.php?title=The_best_strategies_for_Wordle,_part_2&oldid=58"

