Fractal Word Search: How Deep to Delve

K. Chura and T. Khovanova

Abstract - We look at the puzzle *In the Details*, which appeared in the 2013 MIT Mystery Hunt and which gained fame as the *fractal word search*. This seemingly impossible puzzle, whose solution could not fit the memory of a modern computer if the puzzle were solved using a brute-force approach, requires an understanding of its fundamental structure to be cracked. In this paper, we study fractal word searches in a general setting, where we consider one- and two-dimensional word searches with alphabets of any length and replacement rules of any size. We prove that the puzzle is solvable within a finite number of steps under this generalization and give an explicit upper bound on the latest level on which a word of a given length can appear for the first time in a given direction.

Keywords : fractal word search; algorithmic analysis

Mathematics Subject Classification (2020) **:** 00A08

1 Introduction

1.1 Background

In the 2013 MIT Mystery Hunt, a puzzle titled *In the Details* appeared, authored by Derek Kisman. The puzzle, which can also be found at [1], is written out in Figure 1.

This puzzle has since gained the name *fractal word search*. Let us have a look at it together to discover why!

First, we notice that the puzzle looks like a word search. In a word search, one can find words from a given list in a given grid looking horizontally, vertically, or diagonally in eight directions overall. After all the words have been found and their letters crossed out, the leftover letters in the grid spell out the answer or at least a hint on it. Traditionally, the number of blanks in the final row of the puzzle represents the length of the sought answer, which in the case of our puzzle is a length of 8.

However, this puzzle is not a regular word search since only 6 words from the list can be found in the grid. These are $BOUNDARY$ (diagonal in lower-right quadrant), $HENON$ (vertical in lower-left quadrant), $NEURON$ (diagonal in upper-right quadrant), $RIVER$ (horizontal in lower-right quadrant), $YODAWG$ (diagonal in middle of left half), and $LEVELTWO$ (top row). The absence of the other words means that the puzzle has a hidden secret, but what is this secret? We can observe that the words in the list are related to fractals; this is the first big hint. The second hint is that the grid shows many repeating 2×2 blocks, of which there are exactly 26 kinds; this suggests that each block

TWELEVELTWONSHELMUMUOERAIYRANL
QAPIUNPIQAYDPEPIRPRPKVOYESOYOR
ELRATFDTELDTTFDTBWNLMUTFONYDWJ
PIOYJMHAPIHAJMHAAOORRPJMYDANFC
MUOZCGTFBWIRYDHIRAIRTFNCUENCUE
RPVQUHJMAOHKANJUOYHKJMZKBNZKBN
IRONSHOZGOTFUEELTFOEELUEYDOETF
HKYDPEVQDNJMBNPIJMKVPIBNANKVJM
BWIYNLTFSHHIELTWGOYDONDTYDHIOE
AOESORJMPEJUPIQADNANYDHAANJUKV
SHDTYDRPBWUEBWIYTWIWTFYDMUELMU
PEHAANAJAOBNAOESQAQAJMANRPPIRP
ONTWELBWLMSHELTFUEBWBWLMOZEVHI
YDQAPIAOGIPEPIJMBNAOAOGIVQUNJU
DTCGUEYDRPEVNCIREVIRTWUEUETWON
HAUHBNANAJUNZKHKUNHKQABNBNQAYD
IRUERAMUTFELTWONTFOEOEEYDTNLYD
HKBNOYRPJMPIQAYDJMKVKVHWHAORAN
ELGORPNCTFDTYDSHYDELPKTFOZRACG
PIDNAJZKJMHAANPEANPIDFJMVQOYUH
DTMUWJOETFYDELMUMUGORAONIRDTCG
HARPFCKVJMANPIRPRPDNOYYDHKHAUH

BOUNDARY
BROWNIAN
CAUCHY
CURLICUE
DE RHAM
DIMENSION
ESCAPE
HAUSDORFF
HENON
HILBERT
HURRICANE
ITERATE
JULIA
LEIBNIZ
LEVEL ONE
LEVEL TWO
LEVY DRAGON
LYAPUNOV
MANDELBROT
NEURON
NURNIE
POWER LAW
RAUZY
RIVER
SCALING
SPACE
STRANGE
TAKAGI
TECTONICS
T-SQUARE
WIENER
YO DAWG

- - - - - - - -

Figure 1: In the Details

A	B	C	D	E	F	G	H	I	J	K	L	M
TF	GO	IR	HI	EL	CG	RP	UE	BW	PK	OZ	TW	NL
JM	DN	HK	JU	PI	UH	AJ	BN	AO	DF	VQ	QA	OR

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
SH	ON	RA	WJ	YD	MU	DT	OE	EV	NC	EY	IY	LM
PE	YD	OY	FC	AN	RP	HA	KV	UN	ZK	HW	ES	GI

Figure 2: Matching between letters and two-by-two blocks

LEVELONESSUPYPM
EPATETATIMSAORQ
SKFAICRDPCAWHWH
CONKBAHEAUEHRUA
IYMANDELBROTRDU
NTRGIHIYLLARSES
OLEIZNEAHII ZKVD
TFHRGVWCVCLHHLO
CHPSAELOAUUXTMR
EBGWATRNREJAKPF
TSQUARESSBPOCTF

Figure 3: Level one

represents a letter in the alphabet and that each can be replaced with this letter, yielding a grid smaller by a factor of two in each dimension. Finally, to make things a little easier, there are the words $LEVELTWO$ in the upper left corner of the original grid. Could there be the words $LEVELONE$ on the first line of the smaller grid we are looking for? This happens to be the case, and the matching between letters and two-by-two blocks is as in Figure 2.

That is, level one is represented by the 11×15 grid in Figure 3.

This level contains 18 more words from the list, but multiple words are still missing. Hence, we are motivated to continue to level three and perhaps beyond in a search for the rest. Going back to level two and replacing its letters with blocks, we find an additional three words from the list on level three and one word on level four. However, even level four is already a bit too big for inspection, and since the subsequent levels grow in size exponentially, there is no way we could solve the puzzle in reasonable time using the brute-force approach of repeatedly replacing all letters and looking for words on every vertical, horizontal, and diagonal.

Thus, we need to step back and look at how the puzzle works so that we can come up with a smarter approach. How do fractal puzzles work? How deep can a word be hidden that does not appear on any previous level, and is this affected by the characteristics of the

original grid, such as the size of the alphabet, the lengths of the sides of the replacement rules, or the length of the word we are looking for? Can the search for a word hidden very deep be done in a more optimal way than by writing out all the levels? These questions will be the very focus of our paper.

1.2 Overview

This subsection covers our main results. For precise definitions of any of the notation below, see the preliminaries in Section 2.

First, in Section 3, we gain an intuition behind Kisman's puzzle by considering its equivalent in one dimension, with an n-letter alphabet and b-letter replacement rules. We then define a function $\mathcal{W}_1$ on n, b, and the length $|w|$ of a given word w, and we prove in Theorem 3.13 that $\mathcal{W}_1$ is an upper bound on $F_1(b, n, w)$. Here, $F_1(b, n, w)$ is the *latest depth* of w, a function giving the actual latest level on which w can first appear across all possible setups of level one L_1 and all possible sets of replacement rules R with the parameters n and b.

Having focused on the puzzle in one dimension, we use Section 4 to extend our understanding of the fractal word search to two dimensions. We prove in Theorem 4.8 that the function $\mathcal{W}_1$ defined in Section 3 can also be used as an upper bound for $F_2^{hv}(b, n, w)$, the latest depth of a horizontal or vertical word w. Again, n is the size of the alphabet, b is the length of the side of the square replacement rules, and w is the word in question.

The result we obtain in Theorem 4.8, however, does not extend to diagonal words. Hence, we define another function $\mathcal{W}_2$, which takes as its input the length of the alphabet n, the side-length of the square replacement rules b, and the length $|w|$ of the sought word w. We prove in Theorem 4.19 that this $\mathcal{W}_2$ is an upper bound on $F_2^d(b, n, w)$, where $F_2^d(b, n, w)$ is the latest depth of a specific diagonal word w, given an n-letter alphabet and any set R of $(b \times b)$-letter replacement rules.

The rest of the paper is organized in the following way. In Section 2, we codify our notation and provide any preliminary definitions. In Section 3, we prove Theorem 3.13. In Section 4, we prove Theorems 4.8 and 4.19. Finally, in Section 5, we use our results to solve the puzzle, and in Section 6, we evaluate our findings and discuss potential future topics of exploration.

2 Preliminaries

We consider an alphabet on n letters and use upper-case Latin letters for the letters in the alphabet, e.g. A, B, or C. We then call any string of letters from the alphabet a *word*. For instance, CAT and $ARRRRGGG$ are both words constructed from the letters of an alphabet that contains at least the letters A, C, G, R, and T. We use an asterisk $*$ to identify an unknown letter in a word of a known length, e.g., $AT*$ could stand for ATA or ATT or any other 3-letter word beginning with AT. Given a word w, we denote its length by $|w|$. We also refer to a generalized rectangle of letters as a *block*.

In Kisman's word search, as seen in [2], words in most of the eight possible directions are permissible. However, to make the discussion easier, we will focus only on horizontal words from left to right, vertical words from top to bottom, and diagonal words from the upper left to the lower right, and we will thus use the simplified terms *horizontal*, *vertical*, and *diagonal* to refer to these specific directions. Note that the remaining five directions will be analogous to those just chosen.

In our alphabet, each letter has a *replacement rule* of a given size; in this paper, we restrict ourselves to the study of one-dimensional *grids*, where the letters have one-dimensional b-letter replacement rules and two-dimensional grids, where individual letters have square replacement rules of $(b \times b)$ letters. Starting with a grid of letters, we can replace each letter in this grid according to its replacement rules and thus obtain a new, enlarged grid. We can formalize this notion by defining a projection $R(p)$, which maps a block of letters p to its replacement based on the replacement rules of the individual letters of p. We refer to the starting grid as the 1-st *level* or L_1 and denote any consecutive k-th level as L_k, and $R(p)$ can be seen as a mapping between the individual levels, $R(L_k) = L_{k+1}$ for $k \geq 1$. We denote multiple applications of R as $R^l(L_k) = L_{k+l}$ for $k, l \geq 1$.

Example 2.1 Take the 3-letter alphabet consisting of the letters A, B, and C, where the replacement rules of these letters are AB, AC, and BB respectively. If L_1 has the single letter A, we get the consecutive levels $L_2 = R(L_1) = R(A) = AB$, $L_3 = R(L_2) = R(AB) = ABAC$, and $L_4 = R(L_3) = R(ABAC) = ABACABBB$. Thus, we can catch a CAB on L_4. The reader may also notice that with this setup, the word CC can never appear on any level other than 1.

We refer to a word v on L_{k-1} as the *parent* of the word w if $R(v)$ contains w and $R(v')$ does not contain w for any sub-word v' of v. In this case, we call w a *child* of v.

Example 2.2 The parent of CAB on L_4 in Example 2.1 is BA on L_3.

Note that $R(p)$, defined above, does not necessarily have a well-defined inverse. Although a block of letters located in a particular place of the grid on a particular level has a unique parent, the same word in a different place might have a different parent.

Example 2.3 Using the alphabet and replacement rules of Example 2.1, BA could be a child of AA, AB, CA, or CB.

The main interest of this article lies in determining the latest level on which a given word w can appear for the first time across all the possible setups of L_1. Notice that any word can appear on level one, for example, we can set $L_1 = w$. However, the interest of our paper is how long we need to search for a given word to be sure that we find it or to know that it never appears.

3 The One-Dimensional Case

In this section, we prove an upper bound on the latest level on which a word can appear for the first time in one dimension, with an n-letter alphabet and b-letter replacement

rules. Many of the results given here are directly relevant to the exploration of vertical, horizontal, and diagonal words in the two-dimensional scenario. We start with definitions.

Definition 3.1 *Given an initial level L_1 and replacement rules R, we define the* depth *of a word w denoted $d_w(L_1, R)$ to be the first level w appears on. If it does not appear, we set $d_w(L_1, R) = -1$.*

Example 3.2 Given the alphabet in Example 2.1, we examine the depth of the word A across different setups of L_1. If L_1 contains A, then $d_{\mathrm{A}} = 1$. If L_1 does not contain A but contains B, then $d_{\mathrm{A}} = 2$. If L_1 has only instances of the letter C, then $d_{\mathrm{A}} = 3$. In the same setup, we see that the depth of the CC is 1, if it appears on the first level, otherwise we have $d_{\mathrm{CC}} = -1$.

We first make an observation that will help us better understand the transition between individual levels. This observation is the first step towards concluding that any fractal word search is finite.

Lemma 3.3 *If the depth of a word w is k, then the depth of any parent of w is no less than $k - 1$. Moreover, at least one of the parents appears on L_{k-1}.*

Proof. For the first part of the statement, we want to show that if a word w appears for the first time on L_k, then any parent of w appears for the first time no earlier than on L_{k-1}. By contradiction, assume a parent p of w appears on L_j, where $j < k - 1$. Then, since w is a child of p, w appears on L_{j+1}. However, since $j + 1 < k$, this contradicts our initial assumption that w appears for the first time on L_k. Thus, the first time a parent of w can appear is no earlier than L_{k-1}. For the second part of the statement, if none of the possible parents of w appears on L_{k-1}, then the word w itself does not appear on L_k. □

The above observation applies to a fixed initial level. We now introduce the notion of the latest depth that does not depend on the initial level.

Definition 3.4 *Given an n-letter alphabet and b-letter replacement rules R in one dimension, we define the* latest depth *of a word w to be the largest d_w across all L_1,*

$$F_1(b, n, w) = \max\{d_w(L_1, R) \mid L_1 \text{ with only letters represented by } R\}.$$

Example 3.5 From Example 2.1, we get $F_1(2, 3, \mathrm{A}) = 3$, and $F_1(2, 3, \mathrm{CC}) = 1$.

With the observation in the previous lemma, we can establish the latest depth of a 1-letter word, whatever the size of the alphabet and even regardless of the dimensionality of the grid and the size of the replacement rules. The following result serves as the base case for the latest first-time appearance of a word of any length.

Lemma 3.6 *Given an n-letter alphabet and a 1-letter word w, the latest depth of w is no greater than n:*

$$F_1(b, n, w) \leq n.$$

Proof. The parent of a single letter is always itself a single letter. When tracing back the single-letter parents of parents of w, we see, using Lemma 3.3, that all of them must be unique. Hence, we can only go back $n-1$ levels before we run out of letters that appear for the first time on any given level. Thus, w appears for the first time on L_n at the latest, else it does not appear at all. □

Example 3.7 In Example 2.1, the size of the alphabet is 3. We have $d_{\mathrm{A}} = 1$, $d_{\mathrm{B}} = 2$, and $d_{\mathrm{C}} = 3$. No new letter appears from L_4 onward since there are no other letters left in the alphabet. Thus, all new 1-letter words have a depth within the bound 3 given by Lemma 3.6.

The case of 1-letter words above is, in fact, special, for 1-letter words are the only words that can never, on any level, appear on the join of two or more replacement rules of letters from the previous level. Thus, 2-letter words represent another important base case, namely that of words that can span multiple blocks given by the replacement rules. Thus, understanding 2-letter words in the one-dimensional scenario does most of the job of understanding words of any length.

Lemma 3.8 *Given an n-letter alphabet and a 2-letter word w, the latest depth of w is no greater than n^2+1:*

$$F_1(b, n, w) \leq n^2 + 1.$$

Proof. Suppose w appears for the first time on L_k. Then either w is part of a replacement rule of some 1-letter word p that appears for the first time on L_{k-1}, and we are solving on L_{k-1} the problem of Lemma 3.6, or w has a parent p with two letters, $|p| = 2$. Then, we are recursively solving finding a two-letter word for the first time on L_{k-1}. Tracing back the parents of parents, assuming none of them is a single letter, we can see that all of them must be unique 2-letter words by Lemma 3.3. Hence, we can only go back n^2-1 levels before we run out of all the possible ordered pairs of n letters, so one additional step back must bring us to L_1. This allows the 2-letter word w to appear on L_{n^2+1} at the latest as claimed. Note that if we snap back to the problem of 1-letter words before exhausting all n^2 possible pairs, assuming that we have so far seen m letters, we have descended by at most m^2 levels and have no more than $n-m$ levels to go before we run out unseen letters and reach L_1. Now since $n > m > 0$, we have $1 < n+m$ and $0 < n-m$, so from $n-m < (n+m)(n-m)$, we get $n-m < n^2-m^2$. Thus, $m^2+n-m < n^2$ implies that a 2-letter word can really appear on L_{n^2+1} at the latest as claimed. □

Example 3.9 In Example 2.1, we point out that the word CC never appears, if level 1 is a 1-letter word. Although one can reach this conclusion intuitively by inspecting the replacement rules, Lemma 3.8 tells us that even with the brute-force approach of writing out all the levels, it is enough to check $3^2+1 = 10$ levels since 10 is an upper bound on the latest depth of any 2-letter word. Given that L_{10} in this setup has a mere 512 letters or approximately 7 lines, it would not be that difficult to verify!

The key to figuring out an upper bound on the latest depth for a word of length greater than 2 is determining in how many levels this word can be reduced to a 2-letter

word. Intuitively, while transitioning between levels, the length of the parent on L_{k-1} of the child on L_k will be smaller by a quotient of b, so if the parent of w is p, we have $|p| \approx \frac{|w|}{b}$. However, as the following example shows, this is really only an approximation.

Example 3.10 In Example 2.1, the word AB on L_3 has the children ABA, BAC, and $ABAC$ on L_4. The parent of $ABBB$ on L_4 is the word AC on L_3, while the parent of the overlapping word $CABB$ on L_4 is the word BAC on L_3.

Therefore, it is useful to establish an upper bound on the length of the parent of any word w. The intuition behind focusing on the upper bound is that the longer the parents are along the way of going down levels, the more levels it takes before our initial word reduces to a 2-letter word, which in turn means that w might be hidden deeper.

Lemma 3.11 *Given an n-letter alphabet with b-letter replacement rules and a word w on L_k, an upper bound on the length $|p|$ of a parent p of this word on L_{k-1} is given by*

$$|p| \leq \left\lceil \frac{|w| + b - 1}{b} \right\rceil.$$

Proof. Fix a length ℓ and consider all the possible lengths $|w|$ of a word w for which the maximum length of the parent is ℓ. The parent p of length ℓ on L_{k-1} is replaced by an ℓb-letter string. For w on L_k to at least partially cover each b-letter section of this ℓb-letter string, so that its parent can have length ℓ, we have

$$(\ell - 2)b + 1 < |w|,$$

from which we get

$$(\ell - 1)b < |w| + b - 1,$$

and by dividing on both sides, we have

$$\ell - 1 < \frac{|w| + b - 1}{b},$$

from which the lemma follows. □

Using this and previous results, we can finally establish an upper bound on the function $F_1(b, n, w)$ defined in Section 2, which gives the actual latest depth of a specific word w given an n-letter alphabet and b-letter replacement rules across all possible configurations of L_1. First, we define the function $\mathcal{W}_1$, which is a candidate for the upper bound on F_1. Note below the subtle difference between the parameters of the functions $F_1(b, n, w)$ and $\mathcal{W}_1(b, n, |w|)$. While F_1 provides the latest depth for a *specific* word w, i.e., the output may be different for two words of the same length, $\mathcal{W}_1$ provides the same estimate for all the words of equal length regardless of the letters the words are composed of.

Definition 3.12 *Let $\mathcal{W}_1 : \mathbb{N}^3 \mapsto \mathbb{N}$ be a map such that*

$$\mathcal{W}_1(b, n, |w|) = \begin{cases} n & |w| = 1, \\ \lceil \log_b(b|w| - b) \rceil + n^2 & |w| > 1. \end{cases}$$

Next, we prove that $\mathcal{W}_1$ actually is an upper bound of F_1. We iteratively reduce the word to its largest possible parent, and once we arrive at a 2-letter word, we already know what to do by Lemma 3.8.

Theorem 3.13 *Given a one-dimensional grid, an n-letter alphabet with b-letter replacement rules, and a word w, an upper bound on $F_1(b, n, w)$ is given by $\mathcal{W}_1(b, n, |w|)$:*

$$F_1(b, n, w) \leq \mathcal{W}_1(b, n, |w|).$$

Proof. By Lemma 3.6, we have $F_1(b, n, w) \leq n = \mathcal{W}_1(b, n, 1)$ for $|w| = 1$. By Lemma 3.8, we have $F_1(b, n, w) \leq n^2 + 1 = \mathcal{W}_1(b, n, 2)$ for $|w| = 2$.

Now we use strong mathematical induction on the length of the word w. Assume that there exists a $c \in \mathbb{N}_0$ such that for all words v such that $|v| \leq b^c + 1$, the statement holds, and suppose that we have a word w such that

$$b^c + 1 < |w| \leq b^{c+1} + 1.$$

Consider the maximum size of the parent p of w. By Lemma 3.11, this size does not exceed

$$\left\lceil \frac{|w| + b - 1}{b} \right\rceil \leq \left\lceil \frac{b^{c+1} + 1 + b - 1}{b} \right\rceil = b^c + 1.$$

Thus, by the induction assumption, we know that the parent p of w can appear for the first time no later than $\mathcal{W}_1(b, n, b^c+1) = \lceil \log_b(b(b^c+1)) - b) \rceil + n^2 = c+1+n^2$. Thus, the word w itself appears for the first time no later than $c + 2 + n^2 = \mathcal{W}_1(b, n, b^{c+1} + 1)$ as claimed and $F_1(b, n, w) \leq \mathcal{W}_1(b, n, b^{c+1} + 1) = \mathcal{W}_1(b, n, |w|)$ for all $|w| > 2$. □

Example 3.14 Considering the alphabet and the replacement rules from Example 2.1, we want to find the latest depth of the word $CACABA$.

Looking at the replacement rules AB, AC, and BB for the letters A, B, and C respectively, we see that if the word $CACABA$ appears later than L_1, its parent must be either $BBAA$ or $BBAB$ on L_{k-1}. If the parent is $BBAA$, then L_{k-1} is L_1 since we cannot have two consecutive letters A on any level but the first. If, however, the parent is $BBAB$, it could have the parent CA on L_{k-2}. We continue like this, considering all the possible parents of parents, and stop whenever one of two conditions is met. First, if we reach a word that cannot be a child of any other word, then this word can only be found on L_1. Second, if we reach a word on a hypothetical L_{k-x} that has already been seen on a level between L_{k-x} and L_k, we can ignore it. This approach can be visualized using a graph like in Figure 4, where the leaf nodes represent the stopping words in the search.

Hence, we conclude that L_{k-5} is the deepest L_1 we can reach. If we start with level L_1 being A, the next few levels are AB, $ABAC$, $ABACABBB$, $ABACABBBABACACAC$, followed by L_6,

$$ABACABBBABACACACABACABBBABBBABBB,$$

with $CACABA$ on it. Hence, the latest depth for the word $CACABA$ is 6. This satisfies the constraint given by Theorem 3.13, which suggests the upper bound $\lceil \log_2(10) \rceil + 9 = 4 + 9 = 13$.

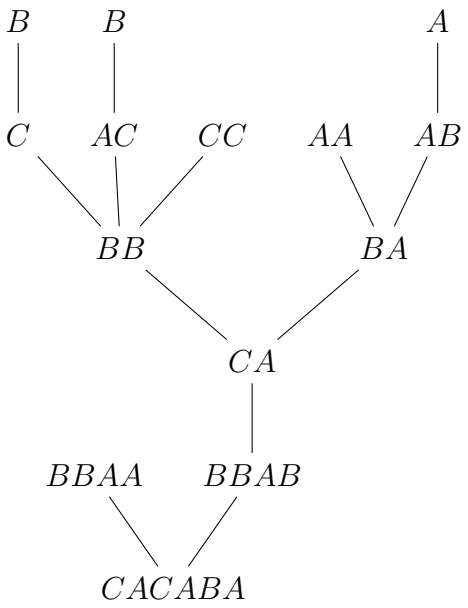

Figure 4: Possible lineage for the word $CACABA$

In the example above, the latest depth of the word $CACABA$ is much lower than the upper bound suggested by Theorem 3.13. This is a more general observation.

Example 3.15 Using computer-aided verification, one can show that given 2-letter replacement rules for a 2-, 3-, and 4-letter alphabet, the actual latest depths of any word are $4 < 2^2 + 1$, $7 < 3^2 + 1$, and $13 < 4^2 + 1$ respectively[1].

To conclude, in this section, we have proven an upper bound on the latest depth of a word of any length in one dimension, with an n-letter alphabet and b-letter replacement rules, and we can now apply all our findings to the two-dimensional scenario.

4 The Two-Dimensional Case

In this section, we build on the results from the previous section and start with expanding our definitions to two dimensions.

4.1 Definitions

We will see later that diagonals behave differently than horizontal and vertical directions. Because of that, we want to have a separate definition for the diagonal depth.

Definition 4.1 *Given an initial level L_1 and replacement rules R in two dimensions, we define the* horizontal/vertical depth *of a word w denoted $d_w^{hv}(L_1, R)$ and* diagonal depth

[1]Google Colab Notebook

of a word w denoted $d_w^d(L_1, R)$ to be the first level w appears horizontally or vertically and diagonally, respectively. If it does not appear, we set the corresponding parameter to -1.

Definition 4.2 *Given an n-letter alphabet and b-letter replacement rules R in two dimensions, we define the* horizontal/vertical latest depth *of a word w to be the largest d_w^{hv} across all L_1,*

$$F_2^{hv}(b, n, w) = \max\{d_w^{hv}(L_1, R) \mid L_1 \text{ with only letters represented by } R\}.$$

We define the diagonal latest depth *of a word w to be the largest d_w^d across all L_1 and R,,*

$$F_2^d(b, n, w) = \max\{d_w^d(L_1, R) \mid L_1 \text{ with only letters represented by } R\}.$$

The implication of the following subsections is that the search in Kisman's puzzle is finite. We start off by defining a useful tool that simplifies the study of diagonal words and also helps collapse the problem of vertical and horizontal words into the already solved problem in one dimension.

Now, we define the concept of a *bounding box*. The motivation behind this comes from several observations about diagonal words and their parents. We first notice that if a word w is diagonal, its parent p might not be a diagonal word. In Figure 5, the parent of the left word can be represented as three letters in a shape of L, $\begin{smallmatrix} * & \\ * & * \end{smallmatrix}$, while the parent of the word on the right are three letters in the shape of inverted L, $\begin{smallmatrix} * & * \\ * & \end{smallmatrix}$. Hence, we may need to discuss not only words but also sets of letters.

Figure 5: Possible 3-letter diagonal words with 3-letter parents

Second, the parent of a diagonal word might have just as many letters as the word itself, as demonstrated in the example below. Therefore, we require a tool that captures how exactly the parent is smaller.

Example 4.3 Consider the 3-letter alphabet A, B, C with (2×2)-letter replacement rules, where the letters A, B, and C are replaced by the blocks $\begin{smallmatrix} A & B \\ C & B \end{smallmatrix}$, $\begin{smallmatrix} A & C \\ B & B \end{smallmatrix}$, and $\begin{smallmatrix} B & B \\ C & C \end{smallmatrix}$, respectively. Then the diagonal word $ABAB$ could have the diagonal parent AB, and the diagonal word $CBBC$ could have the following parent of width 2 and height 3,

A*
CB
*B.

As we will see, the bounding box accomplishes both the goal of tracking sets of letters and of capturing the size of a parent in a meaningful way.

Definition 4.4 *Let the bounding box of a given set of letters in a grid be the smallest rectangle that contains the set.*

Example 4.5 The bounding box of the set of letters

AB*
**B
B

in the two-dimensional grid

ABAC
CBBB
BBAC
CCBB

is the following (3×3)-letter box

ABA
CBB
BBA.

Following the above definition, we observe that the width and height of the bounding box of a set of letters w are independent of each other when tracing back the parents of w. Thus, we conclude that each dimension of the bounding box of a parent p to a set of letters w behaves according to Lemma 3.11.

Lemma 4.6 *Given a two-dimensional grid, an n-letter alphabet with $(b \times b)$-letter replacement rules, and a set of letters w with an $(y \times x)$-letter bounding box on L_k, each side of the box shrinks on L_{k-1} according to the bound established for words in the one-dimensional scenario.*

Proof. Consider the width x of the bounding box. The reasoning for the height y is analogous. We observe that the maximum width of the parent p of the bounding box is constrained by x and the width b of the replacement rules the same way the maximum length of a parent is constrained by the child's length and the length of the replacement rules in the one-dimensional scenario by the proof of Lemma 3.11. □

Example 4.7 In Example 4.3, both examples satisfy the bound $\left\lceil \frac{4+2-1}{2} \right\rceil = 3$ on the length of each of the parent's sides. Notice that for $CBBC$, the width and height of its parent are not equal.

Therefore, thanks to the bounding box, we do not need to worry about whether the parent of w is a diagonal word or a zig-zag formation of letters. Instead, we are concerned about the width and height of the formation. Since each of the sides of the bounding box shrinks with the same speed as a one-dimensional grid, we are left to investigate what happens if the box reduces to a (2×2)-letter formation. However, before we do so, we note how the concept of the bounding box collapses the problem of vertical and horizontal words to the already solved problem in one dimension.

4.2 Vertical and Horizontal Words

In this subsection, we directly apply the notion of a bounding box to horizontal and vertical words in the two-dimensional scenario. The following result should not be surprising since, intuitively, rows and columns of the two-dimensional grid themselves behave like one-dimensional grids.

Theorem 4.8 *Given a two-dimensional grid and an n-letter alphabet with $(b \times b)$-letter replacement rules, a vertical or horizontal word w shrinks like the same word in the one-dimensional scenario, that is*

$$F_2^{hv}(b, n, w) \leq \mathcal{W}_1(b, n, |w|).$$

Proof. Without loss of generality, consider a horizontal word w since the proof for a vertical word is analogous. The bounding box of w is the word itself, i.e., it has $1 \times |w|$ letters. We notice that the vertical side of the box is already as small as possible and cannot get larger. By Lemma 4.6, the horizontal side shrinks according to the bound given by Lemma 3.11 until it reaches a length of 2. Hence, we have reduced the problem to that of a horizontal 2-letter word. Since a horizontal 2-letter word can span at most two horizontally neighboring replacement rules and can therefore only have a 1-letter or horizontal 2-letter parent, Lemma 3.8 can be extended to this situation, and we reach L_1 after no more than n^2 levels. Hence, the reasoning of Theorem 3.13 applies, and $\mathcal{W}_1(b, n, |w|)$ is an upper bound on $F_2^{hv}(b, n, w)$. □

Example 4.9 Consider the same alphabet and replacement rules as in Example 4.7. Like in Example 2.1, we notice that if L_1 does not contain the horizontal or vertical word AA, this word never appears horizontally or vertically. Again, one could reach this conclusion intuitively, but Theorem 4.8 tells us that even with the brute-force approach of writing out all the levels, it would be enough to check $3^2 + 1 = 10$ levels since 10 is an upper bound on the latest depth of any horizontal or vertical 2-letter word.

Remark 4.10 Since in a two-dimensional grid, the number of rows/columns grows exponentially with each level, we expect it to be generally faster to find any given word on a horizontal/vertical than to find the same word in a growing one-dimensional grid, i.e., we expect to find the word on an earlier level in two dimensions. However, note that the function $\mathcal{W}_1(b, n, |w|)$ introduced in Definition 3.12 is just as tight an upper bound on $F_2^{hv}(b, n, w)$ as it is on $F_1(b, n, w)$ since our replacement rules could each have all its

rows identical to each other, in which case all rows on any level are identical, and our two-dimensional case is isomorphic to the one-dimensional case.

Having established an upper bound on $F_2^{hv}(b, n, w)$ or the latest level on which a horizontal or vertical word w can appear for the first time, we can now apply the concept of a bounding box to the scenario by which it was motivated, i.e., diagonal words.

4.3 Diagonal Words

Thus, we have shown that horizontal and vertical words in two dimensions differ very little from words in a one-dimensional grid. Once we focus on diagonal words of length greater than 1, however, the dynamic changes. First of all, the following lemma demonstrates that a diagonal 2-letter word may be found for the first time at a later level than a horizontal or vertical 2-letter word. This is due to the fact that a diagonal 2-letter word can have horizontal, vertical, or diagonal 2-letter parents.

Lemma 4.11 *Given a two-dimensional grid, an n-letter alphabet, and a diagonal 2-letter word w, the latest depth of w is no greater than $2n^2 + 1$:*

$$F_2^d(b, n, w) \leq 2n^2 + 1.$$

Proof. Suppose w appears for the first time on a diagonal on L_k. If w is contained in a replacement rule of some 1-letter word p that appears for the first time on L_{k-1}, we are solving on L_{k-1} the problem of Lemma 3.6, else w has a parent p with two letters. Then, two possible scenarios arise. Either p is vertical or horizontal and L_{k-1} is at most L_{n^2+1} by Theorem 4.8, so L_k is at most L_{n^2+2}. Or p is diagonal, and we are recursively solving finding a diagonal 2-letter word for the first time on L_{k-1}. Tracing back the parents of the parents, assuming none of them is a single letter, we can see all of them must be unique diagonal 2-letter words. Hence, we can only go back $n^2 - 1$ levels before we run out of all the possible ordered pairs of n letters on the diagonal, so one additional step back must bring us to L_1 or to a vertical or horizontal 2-letter word. At most n^2 levels of diagonal 2-letter words added to at most n^2 levels of vertical or horizontal words allows the diagonal 2-letter word w to appear on L_{2n^2+1} at the latest as claimed. □

Example 4.12 Consider the alphabet and replacement rules from Example 4.7 and the single letter A on L_1. Then the first three levels are

A,

AB
CB,

ABAC
CBBB
BBAC
CCBB.

The depth of the diagonal word BB is 3. This satisfies the constraint given by Lemma 4.11, which sets the upper bound on the latest depth of a diagonal 2-letter word as $18+1=19$. Notice also that the diagonal word AA can never appear. Although this conclusion can be reached intuitively, one could go through all the 19 levels to verify the same result manually.

Example 4.13 Consider the alphabet and replacement rules from Example 4.7. Then, if the word AC does not appear on L_1, it cannot appear later on the vertical or the horizontal, but it can appear later on the diagonal.

Remark 4.14 The upper bound given by Lemma 4.11 is very lenient, for it completely disregards the fact that while tracing back all the diagonal 2-letter words, we will have seen many of all the possible vertical and horizontal 2-letter words.

We already showed that a parent of a diagonal word might not be a word, meaning that its letters might not form a line. But what shapes are possible? The following lemma explains.

Lemma 4.15 *Any ancestor of a diagonal word is contained in two neighboring diagonals.*

Proof. Consider letters A and B that have at least one diagonal in between. After applying a (2×2)-letter rule, consider letters X and Y that belong to the child blocks of A and B. We can see that they have to be separated by at least one diagonal. Continuing further, we cannot get to a diagonal word, which completes the proof. □

Corollary 4.16 *If a word's ancestor is contained in a (2×2)-letter bounding box, it cannot have more than 3 letters, and if it has exactly 3 letters, it has to be one of the shapes $\begin{smallmatrix} * & \\ * & * \end{smallmatrix}$ or $\begin{smallmatrix} * & * \\ & * \end{smallmatrix}$.*

Imagine now a situation like in Figure 5 where the parent of a diagonal 3-letter word is confined in a (2×2)-letter box while not being a word itself. As the following lemma demonstrates, such 3-letter formations have a behavior of their own when we trace their parents.

Lemma 4.17 *Given a two-dimensional grid, an n-letter alphabet, and a 3-letter set of letters w inside a (2×2)-letter box, the latest depth of w is no greater than n^3+n^2+1:*

$$F_2^d(b,n,w)\le n^3+n^2+1.$$

Proof. For the two possible configurations of w, see Figure 5. Suppose w appears for the first time on L_k. If w is part of a replacement rule of some 1-letter word p that appears for the first time on L_{k-1}, we are solving on L_{k-1} the problem of Lemma 3.6. If the parent of w is a 2-letter word, necessarily horizontal or vertical, the problem reduces to that of Theorem 4.8. There is, however, one more possible configuration, namely that of a 3-letter parent p contained within a (2×2)-letter block and shaped as an L with the same rotation as w. Parallel to the proof of Lemma 3.8, we can see that this yields a

maximum of $n^3 - 1$ levels before we run out of ordered sets of three letters and reduce to a 1-letter word or 2-letter vertical or horizontal word. Afterwards, we have at most n^2 levels before we descend to L_1. Hence, the latest level on which a 3-letter set of letters w inside a (2×2)-letter box can appear for the first time is $n^3 + n^2 + 1$. □

We now have all the building blocks in hand and can finally formulate a theorem for an upper bound on the latest depth of any diagonal word in a two-dimensional grid. We define a function that is an upper bound on the sought $F_2^d(b, n, w)$.

Definition 4.18 *Let $\mathcal{W}_2 : \mathbb{N}^3 \mapsto \mathbb{N}$ be a map such that*

$$\mathcal{W}_2(b, n, |w|) = \begin{cases} n & |w| = 1, \\ 2n^2 + 1 & |w| = 2 \\ \lceil \log_b(b|w| - b) \rceil + n^2 + n^3 & |w| > 2. \end{cases}$$

Theorem 4.19 *Given a two-dimensional grid, an n-letter alphabet with $(b \times b)$-letter replacement rules, and a diagonal word w, an upper bound on $F_2^d(b, n, w)$ is given by $\mathcal{W}_2(b, n, |w|)$:*

$$F_2^d(b, n, w) \leq \mathcal{W}_2(b, n, |w|).$$

Proof. By Lemmas 3.6 and 4.11, we have $F_2^d(b, n, w) \leq n = \mathcal{W}_2(b, n, 1)$ and $F_2^d(b, n, w) \leq n^2 + 1 = \mathcal{W}_2(b, n, 2)$ as feasible upper bounds on the latest levels on which the diagonal words of lengths 1 and 2 can appear.

The key to proving the case for $|w| > 2$ is understanding that unlike in the one-dimensional scenario and unlike in the two-dimensional scenario for vertical and horizontal words, tracing back the parents of parents of a longer diagonal word may never reduce to the problem of a 2-letter diagonal word. However, we know by Lemma 4.6 and Corollary 4.16 that we can at least reduce any diagonal word w to a parent contained in a (2×2)-letter block on some previous level and that this parent will have no more than 3 letters.

Now, since both the vertical and horizontal dimensions of the parents of parents of a diagonal word w reduce the same way as the length of a word in the one-dimensional scenario does and since the end goal is 2 letters in each dimension, we can directly apply the reasoning from the proof of Theorem 3.13 and conclude that the maximum number of levels required to reach the (2×2)-letter block is given by $\lceil \log_b(b|w| - b) \rceil$. However, as observed, the parent contained in this block may have 3 letters to it. Therefore, after reaching this block, by Lemma 4.17, there will still be up to $n^3 + n^2 + 1$ levels left until definitively reaching L_1. □

Remark 4.20 After this work was released, Derek Kisman wrote a program showing that our bound is fairly tight [3]. Specifically, he created a configuration that puts a 3-letter word at depth $\text{lcm}(a, b, c) + 1$, where $a, b, c \leq n - 3$, and lcm is the least common multiple. If n is even, the lower bound is $(n - 3)(n - 4)(n - 5) + 1$. If n is odd, the lower bound is $(n - 4)(n - 5)(n - 6) + 1$. Asymptotically, these are close to our upper bound.

5 Solution to *In the Details*

We now have the understanding we need to be able to return to the puzzle solution and connect everything!

Using the language introduced for generalization, the puzzle corresponds to an alphabet of size 26 and replacement rules of length $b = 2$. In Section 1, we last found some words on L_4 and noted that writing out every letter on every level would not be an optimal strategy. Indeed, if we reveal that the word $RAUZY$ is hidden furthest on L_{86} and realize that the number of letters in the grid on this level is $11 \times 15 \times 4^{85}$, a number with 54 digits, we can see that this level alone would require 10^{42} terabyte thumb drives just for storage. For reference, at the time of writing, the world's largest single-memory computing system consists of a little over 100 terabytes.

A more productive way is to solve the puzzle backwards by looking for the possible parents of the missing words, similar to what we did in Example 3.14. As our analysis shows, very fast, we get a potential parent contained in a (2×2)-letter bounding box and consisting of not more than 3 letters. We also showed that such a parent cannot have more than $26^3 = 17576$ potential 3-letter parents. This makes searches for all possible parents doable by a program. In reality, the searches are much faster than our bound.

As recalled in the introduction, in some word searches, when one crosses out all the words on the list, one can read a secret message from the remaining letters in the grid. Could this be the case with our puzzle, too? Yes! We can cross out on L_1 the words that it contains, as well as those letters whose descendants become parts of the other words from the list located on other levels. Even before we can find all the words from the provided list, the number of remaining letters on level one can be reduced enough for it to be possible to read the secret message: "SUM EACH WORD'S LEVEL. X MARKS SPOT." This implies that the redundancy of the English language is not enough to solve the puzzle, and we do need to find all the words.

Once we discover the locations of all the words in the search, we can sum the earliest levels on which these words are found, and this sum will determine the level containing the solution to the puzzle. As the secret message suggests, the solution will be an eventual replacement of the letter X, which is not part of any replacement rule and, therefore, only appears once on L_1.

Keep in mind that our newly acquired intuition tells us that diagonal words may potentially be hidden later in the puzzle than horizontal and vertical words. Indeed, we can see that horizontal grouping RQ expands to a horizontal word $LEVYDRAGON$ on L_6. Three more diagonal words $ESCAPE$, $DIMENSION$, and $RAUZY$ appear on L_{15}, L_{17}, and L_{86} respectively. We see that, as we expected, diagonal words appear on much later levels in this puzzle than horizontal words.

Finally, by summing the levels of all found words, we determine the level at which to find the solution. And indeed, the 8-letter answer $HUMPHREY$ appears in the center of level 167 and is appropriately in the shape of an X,

H**Y
UE

RM
H**P.

6 Conclusion

Fractals, a field of study dating back to the 1980s when it was invented by Benoit B. Mandelbrot in [4], has since become widely popular, with applications in biology, technology, and beyond. However, the appearance of fractals in a word search puzzle could definitely be considered one of a kind.

An additional parallel to the puzzle studied is the Thue-Morse sequence, which, using our terminology, corresponds to the one-dimensional grid with a 2-letter alphabet and 2-letter replacement rules. Suppose the alphabet is $\{0, 1\}$, and the rules replace 0 with 01 and 1 with 10. Then, if we start with 0 on L_1, we get the Thue-Morse sequence as a limiting sequence. A discussion of the Thue-Morse sequence can be found in [5]. Hence, the problem we have solved in this paper is not an isolated one but rather one showcasing several areas of mathematics.

We have proven an upper bound on the latest depth of a word w or the latest level on which it can appear for the first time, given the sizes of the alphabet and the replacement rules, and we have noted that this upper bound is not tight. We have extended our knowledge from the one-dimensional to the two-dimensional scenario and proven two separate upper bounds, one on the latest depth of a vertical or horizontal word and one on the latest depth of a diagonal word. One future area of exploration would be to determine the actual latest depth of any word in the one-dimensional or two-dimensional scenario. One could also consider higher-dimensional fractal word searches by generalizing the proposed functions to $\mathcal{W}_d(n, b, |w|)$ for any dimension d. Moreover, the behavior of the collection $\{\mathcal{W}_d\}_{d\geq 1}$ could be studied. Finally, we have used our acquired intuition to provide a solution to Kisman's puzzle.

Acknowledgments

The second author thanks Tom Rokicki for encouraging her to look into this topic. Both authors thank Derek Kisman for an exciting puzzle and for programmatically demonstrating the relative tightness of the bounds proposed.

References

[1] D. Kisman, *In the Details*, available online at the URL: `https://puzzles.mit.edu/2013/coinheist.com/get_smart/in_the_details/index.html` (2013).

[2] D. Kisman, *In the Details (Solution)*, available online at the URL: `https://puzzles.mit.edu/2013/coinheist.com/get_smart/in_the_details/answer/index.html` (2013).

[3] D. Kisman, *FractalWordSearchDepth*, available online on GitHub at the URL: `https://github.com/SnapDragon64/Misc/blob/main/FractalWordSearchDepth.cc` (2024).

[4] B.B. Mandelbrot, *The fractal geometry of nature*, W.H. Freeman and Co., San Francisco, Calif, 1982.

[5] J.P. Allouche, J. Shallit, The ubiquitous Prouhet-Thue-Morse sequence, *SETA*, **'98** (1999), 1–16.

Klara Chura
Tufts University
389 Boston Ave
Medford, MA 02155
E-mail: klara.chura@tufts.edu

Tanya Khovanova
Massachusetts Institute of Technology
Room 2-231C
E-mail: tanya@math.mit.edu

Received: May 13, 2024 **Accepted**: December 9, 2024
Communicated by Cynthia Sanchez Tapia