The evaluation of slot grammars and their application in the study of the Voynich Manuscript[1]

Mauro Lanzini[2]

Independent researcher

Abstract

I review the metrics used to evaluate slot grammars, showing the normally used efficiency and F1 score to be fundamentally flawed. After introducing the concept of 'looping' slot grammars, a generalization of standard grammars, I show how any grammar can be used in a distinctive lossless data compression algorithm to generate a compressed Voynich Manuscript text. This allows the definition of a new metric free from fundamental flaws: N_{bits}, the total number of bits needed to store the compressed text. I then compare published state-of-the-art grammars and the newly introduced LOOP-L grammar class using the N_{bits} metric.

Keywords

Voynich Manuscript, Word structure, Slot grammars, Lossless data compression

Introduction

The Voynich Manuscript (VMS) is a handwritten codex with multicolored illustrations, consistently dated to the beginnings of the 15th century [1]. It is in an unknown script which, notwithstanding numerous studies, has constantly defied interpretation. The word types of VMS show a remarkably regular structure, which has nonetheless been impossible to pin down satisfactorily. Recently, 'slot grammars' (or simply 'grammars' from now on) have emerged as a promising tool for the study of VMS word structure [2]. Two recent examples are ThomasCoon's (TC) grammars[3] 2016 [2] and Massimiliano Zattera's (MZ) SLOT 2022 [3] (Figure 1, Figure 2).

Two degenerate grammars are particularly interesting, because they trivially generate all word types (Figure 3) The L*CSET (length * character set) grammar has as many slots as the maximum word length, each slot containing the whole character set. The 1*WSET (1 * word types set) grammar has just one slot, containing all the word types as glyphs.

[1] Published on academia.edu, January 2025

[2] Contact via academia.edu

[3] ThomasCoon proposed two different versions. I call 'V2' the one used in this study.

ThomasCoon's V2 grammar

1	2	3	4	5	6	7	8	9	10	11	12
q	s	o a	k t	ch	e	o	r	d s	o	n m	y
d	ch	y ol	p f	sh	ee	a	l	ch t	a	in r	s
Ø	sh	Ø	Ø	Ø	eee	y	Ø	Ø	Ø	iin l	Ø
	Ø				Ø	Ø				Ø	

Ø = null

Zattera's SLOT grammar

1	2	3	4	5	6	7	8	9	10	11	12
q	o	l	t	ch	cth	e	s	o	i	d l	y
s	y	r	k	sh	ckh	ee	d	a	ii	r m	Ø
d	Ø	Ø	p	Ø	cph	eee	Ø	Ø	iii	n	
Ø			f		cfh	Ø			Ø	Ø	
			Ø		Ø						

Ø = null

Figure 1: TC's V2 and MZ's SLOT grammars

Zattera's SLOT generates 'qokaiin':

q	o	--	k	--	--	--	--	a	ii	n	Hit!
1	2	3	4	5	6	7	8	9	10	11	12
q	o	l	t	ch	cth	e	s	o	i	d l	y
s	y	r	k	sh	ckh	ee	d	a	ii	r m	Ø
d	Ø	Ø	p	Ø	cph	eee	Ø	Ø	iii	n	
Ø			f		cfh	Ø			Ø	Ø	
			Ø		Ø						

But cannot generate 'cheky':

--	--	--	--	ch	--	e	--	--	--	--	--	Miss
1	2	3	4	5	6	7	8	9	10	11	12	
q	o	l	t	ch	cth	e	s	o	i	d l	y	
s	y	r	k	sh	ckh	ee	d	a	ii	r m	Ø	
d	Ø	Ø	p	Ø	cph	eee	Ø	Ø	iii	n		
Ø			f		cfh	Ø			Ø	Ø		
			Ø		Ø							

Figure 2: the slot grammar algorithm, with examples of 'hit' and 'miss'

Degenerate L*CSET grammar

1	2	3	...	max word length
a	a	a	a	a
c	c	c	c	c
d	d	d	d	d
...	...	...	...	...
r	r	r	r	r
s	s	s	s	s
y	y	y	y	y
Ø	Ø	Ø	Ø	Ø

Degenerate 1*WSET grammar

1
daiin
ol
aiin
...
teodyteytar
dalkalytam
psholpchcfhdy
teyqokedy

Figure 3: the degenerate grammars

Box 1: definitions

Word token	A word as it appears in the transcription. The same word token can appear multiple times in the text, for instance 'daiin' is a word token occurring 834 times in the whole RF1a-n standard EVA transcription used in this study (RF1a-n has a total of 37913 word tokens, excluding tokens with rare characters, see Methods).	$N_{wordtokens}$
Word type	A unique word as it appears in the dictionary of the transcription. For instance, 'daiin' is a single word type. RF1a-n has a total of 8012 word types (excluding tokens with rare characters as above).	$N_{wordtypes}$
Glyph	A string of characters appearing in the slots of a slot grammar, including the 'null' character. For instance 'q', 'ch', 's' and 'iin' are glyphs of TC's V2 grammar	N_{glyphs}
Hits	The total number of VMS words the grammar can generate.	N_{hits}
Grammar size	The total number of words a grammar can generate. It's given by the product of the total number of glyphs in each slots: $\prod_i^{SLOTS} Nglyphs_i$	$N_{grammarsize}$
Coverage[4]	Usually over word types: $N_{hits}/N_{wordtypes}$	C
Efficiency[5]	$N_{hits}/N_{grammarsize}$	E
F1 score	Derived from C and E, F1 = 2*C*E/(C+E)	F1

[4] Zattera used the term "Recall" instead of "Coverage"
[5] Zattera used the term "Precision" instead of "Efficiency"

The problem of evaluating grammars

Coverage is, obviously, the primary metric of every grammar. However the degenerate L*CSET trivially reaches a coverage of 100%, which makes a stand-alone coverage metric unsound. To overcome this, the efficiency metric, which is very low when calculated on the L*CSET grammar, has been used together with coverage (and the combined F1 score) to rank different grammars [3] . Importantly however, also the efficiency metric trivially reaches 100% with 100% coverage (and a maximal F1 score) by using the 1*WSET grammar, and is thus unsound.

'Looping' grammars

I propose a generalization of slot grammars[6], where each grammar can be repeated multiple times (the repeating block being called a 'grammar chunk'). This makes little sense using efficiency as a metric, because efficiency exponentially decreases at each repetition, but I will show how this generalization naturally fits the new N_{bits} metric[7].

I will also notice 1*WSET, TC's and MZ's grammars can be thought as a looping grammar with just one repetition, while the L*CSET grammar can be thought as having just one slot and as many repetitions as the maximum word type length.

CHUNKzip: a lossless data compression algorithm based on slot grammars

As the common Lempel-Zev algorithms [5], CHUNKzip works by first finding a list of character strings (a list of 'word chunks'). It then assigns a Huffmann bit code[8] to each chunk and compresses the text by replacing each chunk in the text with the bit code. The compressed text can then be decompressed back to the original text, knowing the chunks list (the chunks 'dictionary').

The peculiarities of CHUNKzip, versus for instance pkzip, are (1) it only searches chunks inside each single word token, not inside the whole text; and (2) it is constrained by the grammar in the set of possible chunks it can use. The algorithm is detailed in Figure 4.

[6] Emma May Smith proposed an analogous concept [7]

[7] An additional consideration makes looping grammars attractive. Existing state-of-the-art grammar have a relatively low coverage (~50%, see later for exact data) and they need ad-hoc explanations for entire classes of word types which the grammar cannot generate. For example many word types look to be composed by the concatenation of two simpler word types and are often defined as 'separable'. In the words of Zattera:

> "For example, chockhy appears 18 times in the text; it is a separable word type that can be divided as cho - ckhy, each part being a regular word type appearing in the text 79 and 39 times respectively."

But by allowing Zattera's SLOT grammar to loop twice (effectively doubling its number of slots) all the 'separable' words types are now generated, thus increasing coverage and, importantly, removing the need for an ad-hoc exception.

[8] Huffmann coding [6] assigns to each chunk a variable-length bit code, with shorter codes for the more frequently used chunks. It is used, for instance, by pkzip. It's an optimal encoding and it has the propriety that no code is the prefix of any other code, thus allowing the resulting bit stream to be decoded.

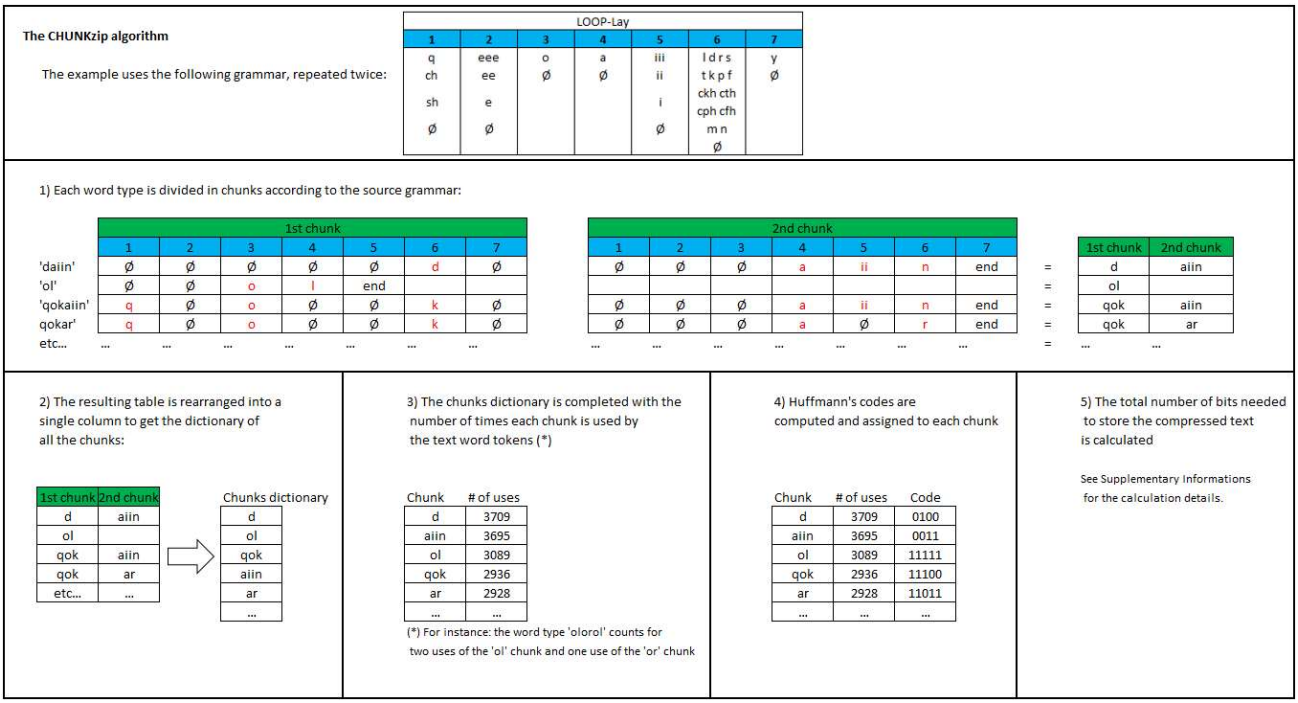

Figure 4: the CHUNKzip algorithm

A new metric for grammars evaluation: N_{bits}

I propose to use the total number of bits need to store the 'CHUNK-zipped' VMS (N_{bits}) as the new metric to replace efficiency for grammars evaluation: a grammar is better the less bits it needs. Intuitively, a better grammar divides the word types in chunks in a more compact way[9].

I computed N_{bits} (see Supplementary data) on the RF1a-n VMS for the two trivial grammars CSET (with 10 repetitions) and WSET, finding N_{bits} = 744114 or 740276, respectively.

Both values are large and this, provided non-trivial grammars score lower values, as we will see is the case, makes the metric sound[10].

The LOOP-L grammar class

Building on the concept of looping grammars, I developed the LOOP-L grammar class. It's a looping grammar based on a simple repeating chunk structure (Figure 5)

[9] The metric N_{bits}/Coverage can also be used, with the advantage of reducing the comparisons to a single number.

[10] It could be argued that considering just one of the possible chunkifications of each word type is a serious drawback of this approach. However, each grammar variation univocally defines a different chunkification, thus the 'chunkification space' can, in principle, be fully explored with different grammars and the N_{bits} metric then gives a way to score each of them.

LOOP-Lay chunk						
1	2	3	4	5	6	7
q ch sh Ø	eee ee e Ø	o Ø	a Ø	iii ii i Ø	l d r s t k p f ckh cth cph cfh m n Ø	y Ø

Figure 5: a typical LOOP-L grammar (LOOP-Lay of Figure 4)

The structure is rather flexible and can be modified easily, creating a 'class' of different variations of LOOP-L (see Figure 6 for examples).

LOOP-La chunk					
1	2	3	4	5	6
q ch sh Ø	eee ee e Ø	o Ø	a Ø	iii ii i Ø	l d r s t k p f ckh cth cph cfh m n y Ø

LOOP-La1 chunk					
1	2	3	4	5	6
q ch sh y Ø	eee ee e Ø	o Ø	a Ø	iii ii i Ø	l d r s t k p f ckh cth cph cfh m n Ø

LOOP-Lb chunk					
1	6	2	3	4	5
q ch sh Ø	l d r s t k p f ckh cth cph cfh m n y Ø	eee ee e Ø	o Ø	a Ø	iii ii i Ø

LOOP-Lc chunk						
1	2	3	4	5	6	7
q ch sh Ø	l d r s t k p f ckh cth cph cfh m n y Ø	eee ee e Ø	o Ø	a Ø	iii ii i Ø	l d r s t k p f ckh cth cph cfh m n y Ø

LOOP-Lx chunk							
1	2	3	4	5	6	6	7
q Ø	o Ø	a Ø	iii ii i Ø	eee ee e Ø	l d r s t k p f ckh cth cph cfh m n Ø	y Ø	ch sh Ø

Figure 6: some possible variations of LOOP-L grammar

LOOP-L grammars reach a very high coverage (>80-90%)[11] starting from four repetitions, while more repetitions only marginally increase it (Figure 7).

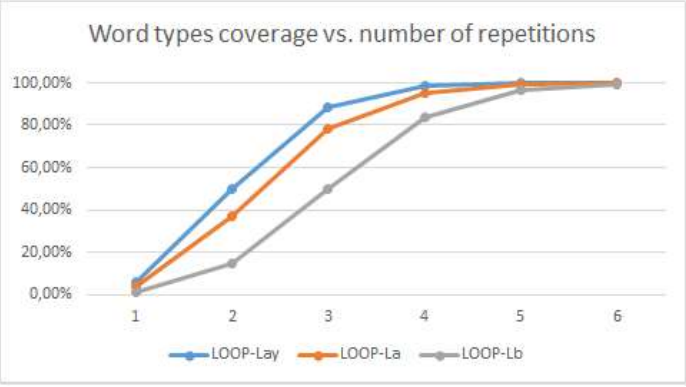

Figure 7: coverage vs. # of repetitions of selected LOOP-L grammars

[11] Coverage gets so high that no ad-hoc exceptions need to be invoked for word types which are not generated by the grammar. For instance only 15 word types are not found by LOOP-Lay with 5 repetitions.

Performances of TC's and MZ's grammars and LOOP-L grammars

From preliminary tests (see Supplementary data), I identified LOOP-Lay and LOOP-La as the most promising according to the N_{bits} metric. I used the metric to evaluate them and TC's and MZ's state-of-the-art grammars. Given TC's and MZ's grammars have a relatively low coverage I compared them with comparable LOOP-L grammars at a low repetition number. I also compared high-coverage, high-repetitions number LOOP-L grammars with 2-repetitions TC's and MZ's grammars, which have similarly high coverages.

Low coverage grammars

Grammar	Repetitions	Coverage	N_{bits}
LOOP-Lay	2	50.0%	355807
MZ's SLOT	1	41.2%	401776
TC's V2	1	54.5%	460118

High coverage grammars

Grammar	Repetitions	Coverage	N_{bits}
LOOP-Lay	5	99.8%	486946
LOOP-La	5	99.2%	491418
MZ's SLOT	2	93.4%	549777
TC's V2	2	82.9%	554531
Trivial CSET	10	98.2%	744114
Trivial WSET	1	100%	740276

The LOOP-Lay grammar compresses the VMS word tokens to the minimum number of bits, which suggests it better captures the structure underlying Voynich words. MZ's and TC's grammars do not fare as well, and the trivial grammars are the worst, as required for the soundness of the N_{bits} metric.

I tested a few variations of LOOP-L grammars (see Supplementary data), always with worse results than LOOP-Lay. I cannot however exclude the existence of better looping grammars (LOOP-L-like or dissimilar)[12].

Limitations of this study

Only intra-word correlations are explored by this study. Short- or long-range correlation between words and the influence of paragraphs and line breaks cannot be modeled. I will however notice that the division of word types in chunks ('chunkification') could be the basis for a 'similarity' metric between words, different from the commonly used metric of graphic similarity, which might be useful in studying short- and long-range correlations.

Conclusions

This study confirms and extends previous results about Voynichese word types conforming to a rigid structure, without the need to invoke any exceptions for 'separated' or 'weird' word types. The concept of looping grammars easily allows reaching very high coverages. The N_{bits} metric is free from fundamental flaws and gives a

[12] Note added in proof: indeed, by combining togeher the 'o' and 'a' slots of LOOP-Lay the resulting grammar (LOOP-Lay-oa) scores better (Coverage = 99.8%, N_{bits} = 484688)

sound way to compare different grammars. Of all the grammars I examined, LOOP-Lay scored the best along this metric (but see Note 12).

Slot grammars can be seen as a way to store information, with each slot able to store $\log_2(N_{glyphs}(slot))$ bits[13]. This could point to a possible way of decoding the Voynich manuscript (if it's meaningful), if it weren't that (1) the possible grammar variations are endless; (2) we have no idea of what kind of data has been encoded; and (3) we have no idea of how the glyphs in the grammar are related to the encoded data. The N_{bits} metric might be useful for tackling point (1).

Methods

The transcription used was the standard reference RF1a-n [8], with metadata and word tokens with an embedded '?' character removed.

Following previous customs, the 'rare' characters 'g', 'x', 'v', 'z', 'b', 'j', 'u' are missing from the grammars and word types including them have not been considered in the counts. Words types containing 'c' or 'h' outside of 'ch', 'sh', 'ckh', 'cth', 'cph', 'cfh' glyphs (for instance 'chckhhy') have not been counted too.

All the computations were made using VoyTextAnalyzer 010 014, a C# program I developed in Microsoft Visual Studio 2022 free edition. VoyTextAnalyzer does more than calculating N_{bits}, it also uses the chunkification process to build a random, asemic generator of pseudo-Voynichese texts which are almost indistinguishable from true VMS on many statistical parameters, including monograms distribution, bigrams distribution, entropies, word tokens distribution (Zipf's law), word tokens length distribution, word types length distributions and word-by-word relative ranking. This may be the subject of a future paper.

Online data

VoyTextAnalyzer 010 014 is available on GitHub: https://github.com/inglanzini/VoyTextAnalyzer-010-014

Additional online data available in the same GitHub repository, Supplementary Data:

- Reference Voynich transcription actually used in this work: "RF1a-n-x7 without question marks.txt"
- Excel dumps of LOOP-Lay grammar, which include the division of each word type in chunks, the chunks dictionary with Huffmann codes and additional data not used in this study but used for generating the pseudo-Voynich asemic text: "Grammar LOOP-Lay 5R.xlsx". For reference, the same data are provided for the two trvial grammars: "Grammar CSET 10R.xlsx" and "Grammar WSET 1R.xlsx".
- An example of an asemically generated pseudo-Voynich: "Pseudo-Voynich asemic seed 0.txt".
- The .exe of VoyTextAnalyzer 010 014: "VoyTextAnalyzer.exe".
- "VoyTextAnalyzer 010 014 User Manual and Software Notes.pdf"
- "TextAnalyzer 010 014 manuale.pdf": user manual (in Italian) of the basic TextAnalyzer software of which VoyTextAnalyzer is an extension.
- A copy of this paper.

[13] For instance 12.9 bits/chunk for LOOP-Lay

Supplementary data

Calculation of N_{bits}

N_{bits} is the sum of the bits needed for the chunks dictionary ($N_{bits\_dictionary}$) plus the bits needed for the compressed text (N_{bits_text}).

The dictionary is a list of chunks, each of which consists in one or more characters taken from the character set of the original text. If C_{size} is the total number of characters used by the chunks then each character requires $log2(C_{size})$ bits. The bits required by the dictionary are then (see Figure 8):

$$N_{bits\_dictionary} = \sum_{chunk}^{Nchun} (log2(C_{size}) * Length(chunk) + Codelength(chunk))$$

The bits required by the compressed text are (see Figure 8):

$$N_{bits\_text} = \sum_{word_{type}=1}^{Nwordtypes} Occurrences(word_{type}) * \sum_{word_{chunk}=1}^{Chunk\ (word_{type})} Codelength(word_{chunk})$$

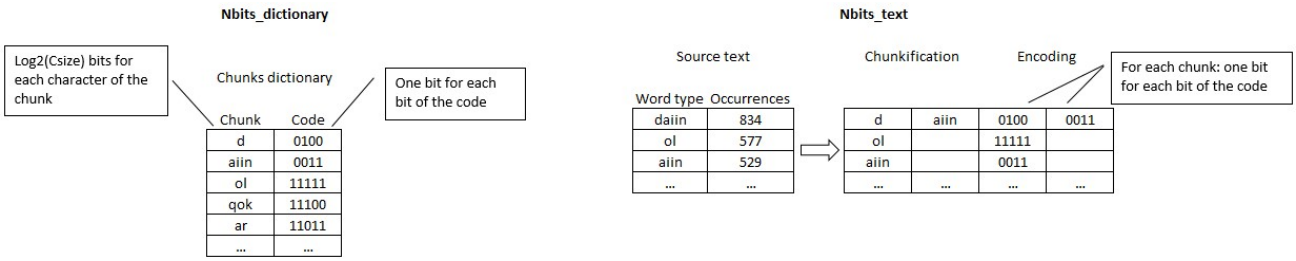

Figure 8: computing N_{bits}

Experiments with loop grammars

I experimented with all the grammars of Figure 6 plus some additional variants along the same scheme (-Lb1, -Lby, -Lc1, -Lcy). I also experimented adding 'lk' and 'ld' glyphs to the ['l', 'd', 'r', 's' etc.] slot. I always obtained a worse N_{bits} than LOOP-Lay (data not shown; LOOP-Lay + 'lk', 'ld' is quite close to LOOP-Lay with N_{bits} = 488415 (but see Note 12).

The 100 most frequent word types divided in chunks by LOOP-Lay

Word type	Occurrences	1st chunk	2nd chunk
daiin	834	d	aiin
ol	577	ol	
aiin	529	aiin	
chedy	517	chedy	
ar	442	ar	
shedy	430	shedy	
or	386	or	
chol	364	chol	
chey	344	chey	
dar	334	d	ar
s	324	s	
qokeey	306	qok	eey
qokeedy	305	qok	eedy
al	299	al	
qokain	274	qok	ain
qokedy	270	qok	edy
shey	268	shey	
qokaiin	259	qok	aiin
dy	244	dy	
dal	239	d	al
chor	214	chor	
okaiin	205	ok	aiin
dain	202	d	ain
qokal	195	qok	al
shol	183	shol	
y	177	y	
cheey	177	cheey	
r	172	r	
okeey	167	ok	eey
otedy	160	ot	edy
cheol	159	cheol	
qokar	157	qok	ar
otar	156	ot	ar

Note: the first 100 words types use at most 2 chunks

Word type	Occurrences	1st chunk	2nd chunk
chy	153	chy	
sheey	152	sheey	
otaiin	152	ot	aiin
okal	152	ok	al
chdy	149	chdy	
qol	146	qol	
oteey	141	ot	eey
qoky	140	qoky	
okar	139	ok	ar
otal	138	ot	al
chckhy	135	chckhy	
saiin	129	s	aiin
okain	126	ok	ain
ain	123	ain	
okedy	122	ok	edy
l	121	l	
sho	120	sho	
oty	116	oty	
oteedy	112	ot	eedy
o	112	o	
lchedy	110	l	chedy
dair	109	d	air
qokey	104	qok	ey
sheol	104	sheol	
oky	103	oky	
cthy	102	cthy	
okeedy	102	ok	eedy
dol	99	d	ol
qokol	98	qok	ol
otain	96	ot	ain
chody	94	chody	
qotedy	93	qot	edy
shor	91	shor	

Word type	Occurrences	1st chunk	2nd chunk
am	89	am	
cheody	87	cheody	
sheedy	86	sheedy	
cheor	85	cheor	
shy	85	shy	
qoty	84	qoty	
sar	84	s	ar
qotaiin	83	qot	aiin
dam	83	d	am
otol	82	ot	ol
chcthy	82	chcthy	
qoteedy	79	qot	eedy
air	78	air	
char	76	char	
cheo	73	cheo	
cho	73	cho	
okol	72	ok	ol
sain	69	s	ain
qotar	68	qot	ar
qotal	66	qot	al
qokchy	66	qok	chy
dor	66	d	or
cheky	65	cheky	
sol	63	s	ol
kaiin	63	k	aiin
odaiin	63	od	aiin
otey	63	ot	ey
okeol	62	ok	eol
qotain	62	qot	ain
qotchy	62	qot	chy
cheedy	61	cheedy	
okey	60	ok	ey
oly	59	oly	
shckhy	59	shckhy	

The 100 most used chunks of LOOP-Lay

Chunk	Times used	Huffmann code
d	3709	0100
aiin	3695	0011
ol	3089	11111
qok	2936	11100
ar	2928	11011
k	2910	11010
al	2685	11000
ok	2401	10100
ot	2391	10011
y	1902	01100
t	1641	00011
ain	1586	00010
l	1550	00000
s	1527	111101
or	1396	110011
chedy	1260	101100
eey	1213	101011
qot	1058	100000
edy	1042	011111
chy	1002	011101
eedy	965	011010
chey	896	010100
dy	836	001001
o	771	000010
chol	764	1111001
ey	755	1110111
am	699	1100101
shedy	669	1011111
p	651	1011101
chdy	560	1001000
air	549	1000101
r	546	1000100
chor	491	0110111

Chunk	Times used	Huffmann code
od	471	0101101
shey	463	0101011
op	375	11101101
cth	370	11101011
ched	324	10111100
eol	322	10111000
cheey	321	10110111
cheol	302	10101010
ody	296	10010111
shy	280	10001111
chd	277	10001101
shol	276	10001100
aly	258	01111011
qol	246	01110001
eody	242	01101101
qo	240	01101100
ed	221	00101100
cho	219	00101011
sheey	208	00100010
f	196	00001110
e	195	00001100
sho	190	111100001
chok	187	111011001
chod	184	111010101
os	176	111010001
eeey	175	110010010
chody	167	101111011
qop	163	101110011
sheol	158	101101100
char	158	101101011
eed	157	101101010
ky	157	101101001
oty	151	101010011

Chunk	Times used	Huffmann code
ckh	151	101010010
qoky	151	101010001
cheo	149	101010000
chckhy	146	100101011
cheor	145	100101001
chk	142	100100111
shor	140	100100100
chot	140	100011101
ary	139	100011100
cheody	136	100001110
oly	131	100001011
aiir	130	100001001
oky	130	100001000
ees	128	011110001
chal	126	011100101
oiin	126	011100100
cph	120	010111011
aiiin	120	010111010
chek	119	010111001
eod	118	010110011
eeody	116	010110001
shdy	116	010110000
sheedy	115	010101011
eor	114	010101010
shed	113	010101000
cthy	110	001011010
om	108	001010011
eeol	108	001010010
cheedy	105	001010000
eo	104	001000110
qod	103	001000001
ak	101	000011110
eeo	99	000011011
an	94	1111000000

The total number of chunks of LOOP-Lay (Nchunktypes) is 766

References

[1] G. Koen, “Is the Voynich manuscript fake?”, 2024, https://www.youtube.com/watch?v=ThGlIQKhByE

[2] René Zandbergen, “Analysis Section (3/5) - Word Structure”, 2024. http://www.voynich.nu/a3_para.html

[3] Massimiliano Zattera, “A New Transliteration Alphabet Brings New Evidence of Word Structure and Multiple "Languages" in the Voynich Manuscript”, 2022. https://ceur-ws.org/Vol-3313/paper10.pdf

[4] Nick Pelling, “Sean Palmer’s Voynichese word generator”. http://ciphermysteries.com/2010/11/22/sean-palmers-voynichese-word-generator

[5] Jacob Ziv; Abraham Lempel, “A universal algorithm for sequential data compression”, 1977. IEEE Transactions on Information Theory. 23 (3): 337-343

[6] David Huffmann, “A method for the construction of minimum-redundancy codes”, 1952. Proceedings of the IRE. 40 (9): 1098-1101

[7] Emma May Smith, “A new word structure”, 2019. https://agnosticvoynich.wordpress.com/2019/05/06/a-new-word-structure/

[8] http://www.voynich.nu/data/RF1a-n.txt

I thank the https://www.voynich.ninja/ community for their help and for the stimulating discussions.