Phaistos-Davis 2018 - Extended

May 2, 2021

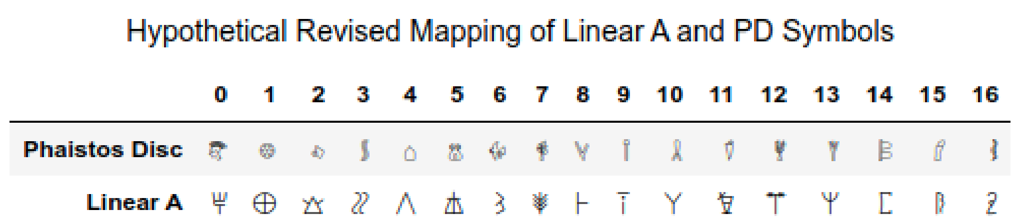
1 Recreating the Results in "The Phaistos Disk: A New Way of Viewing the Language Behind the Script" (Davis 2018)

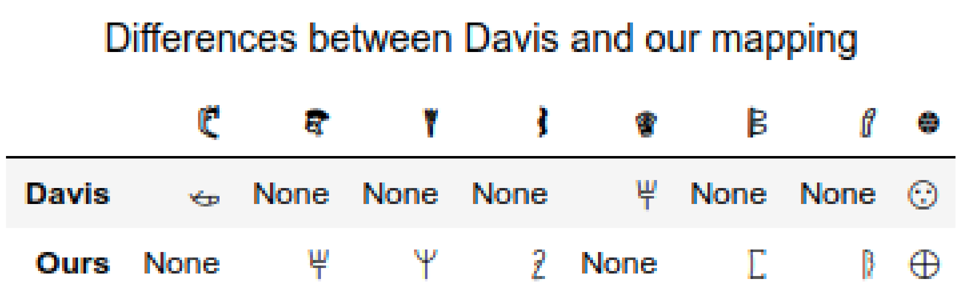
(Note: You will need to install a font to view the Phaistos Disc glyphs used in this article.)

I use the Linear A corpus at https://lineara.xyz to recreate the results from Brent Davis' paper showing a statistically significant relationship between the bigrams in the Phaistos Disc and Linear A.

My findings from this exercise are:

- I find that one bigram identified as common between the two is doubtful. ‘ ’ (TI-I) does not actually appear in Linear A. It may be that the bigram is with a variation of (TI) which is *(28B). We find a single instance of ‘ ’ (TI*-28B) in the Linear A corpus, in ZA6b. It is not clear to me if it is valid to treat (TI-*28B) as the equivalent of (TI-I). If it is not, then the number of matching bi-grams between the Phaistos disc and the Linear A corpus must be revised down to 16. This no longer falls within the region of statistical significance, which Davis identifies as 16.4 or above.
- I get a better p-value than Davis 2018 for his mapping.
- I propose an alternative mapping of PD and Linear A symbols that achieves a better proportion of bigrams found in both Linear A and the Phaistos Disc and a substantially lower p-value than Davis 2018. This mapping is as follows:





1.1 Recreating the Results of Davis 2018

First we import the Phaistos Disc inscription. We also initialize a list of symbols from the Phaistos Disc and all known symbols from Linear A.

[252]:
```
import json
import itertools as it
import pandas as pd
from IPython.display import display
pd.set_option("display.latex.repr", True)

styles = [dict(selector="caption",
    props=[("text-align", "center"),
    ("font-size", "120%"),
    ("color", 'black')])]

pd_inscription_a = ("   |  |  |  |   |   |  |   |  |"
                 "   |   |   | |    | |   |     | |"
                 "   |    | |  |     |   |  |   |  |  |"
                 "    | |  |")
pd_words_a = pd_inscription_a.split('|')

pd_inscription_b = ("   |   |   |  |   |   |   |    |   |"
                     "   |   |   |    |  |  |    |   | "
                     " |  |  |    |   |  |  |   |    |    |"
                     "   |   | |")
pd_words_b = pd_inscription_b.split('|')

pd_inscription = pd_inscription_a + pd_inscription_b
pd_words = pd_inscription.replace('','').split('|')

pd_symbols = [" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",␣
 ↪" ", " ", " ", " ", " "
    , " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " "]

la_symbols = [" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",␣
 ↪" ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
```

```
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
    " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " "]
```

Next we import all known words from Linear A into a list called `la_words`.

[198]:
```python
json_file = open('../Data/LinearAWords.json')
inscriptions = json.load(json_file)

la_words = []
for inscription in inscriptions:
    word_tags = inscription["tagsForWords"]

    for index, word_tag in enumerate(word_tags):
        tags = word_tag["tags"]
        if "word" not in tags:
            continue
        word = word_tag["word"].replace('\U0001076b', '')
        if len(word) == 1:
            continue
        la_words.append(word)
la_words = list(set(la_words))
```

Now we can create lists of unique bigrams in Linear A and the Phaistos disc.

[199]:
```python
def getNgrams(words, n):
    ngrams = []
    for word in words:
        bg = [word[i:i+n] for i in range(0, len(word) - (n-1))]
        ngrams.extend(bg)
    return ngrams

la_bigrams, pd_bigrams, pd_trigrams, la_trigrams = [], [], [], []
ngram_infos = [
    [la_bigrams, "bi", 2, la_words, "Linear A"],
    [pd_bigrams, "bi", 2, pd_words, "Phaistos Disc"],
]

for (ngram, prefix, n, words, name) in ngram_infos:
    ngram = getNgrams(words, n)
```

```
    print("\n" + name + ":")
    print("Unique " + prefix + "grams", len(set(ngram)),
          "Total " + prefix + "grams", len(ngram))
    print("Unique symbols in " + prefix + "grams",
          len(set(list(it.chain.from_iterable(ngram)))))

la_bigrams = getNgrams(la_words, 2)
pd_bigrams = getNgrams(pd_words, 2)
```

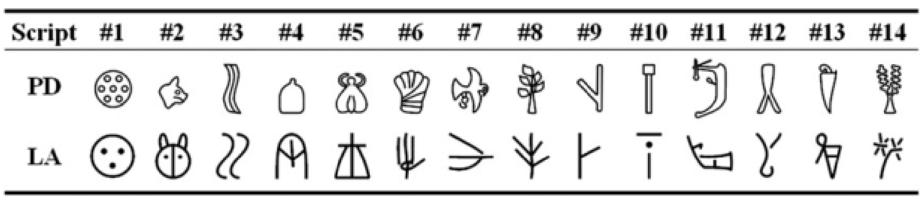
```
Linear A:
Unique bigrams 1170 Total bigrams 2036
Unique symbols in bigrams 168

Phaistos Disc:
Unique bigrams 115 Total bigrams 180
Unique symbols in bigrams 45
```

With these we now have what we need to rerun Davis' analysis comparing the bigrams that appear in both Linear A and the disc.

Davis gives the homomorphs used for his analysis as follows:

TABLE 39

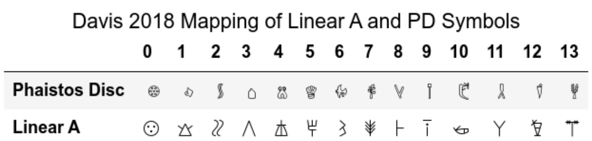
The 14 PD signs from Table 32 that form word-internal pairs on the PD, together with their LA homomorphs



We implement the same here:

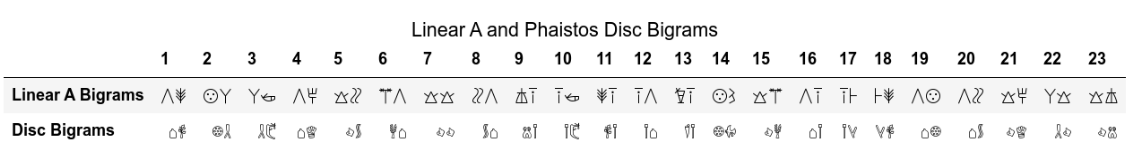
[272]:
```
# Brent Davis 2018 mapping
pd_la_davis_map = {
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
```

```
" ": " ",
}

df = pd.DataFrame([pd_la_davis_map.keys()
                  , pd_la_davis_map.values()
                  ])
df = df.set_axis(["Phaistos Disc", "Linear A"], axis='index')
df = df.style.set_caption("Davis 2018 Mapping of Linear A and PD Symbols").
 ↪set_table_styles(styles)
display(df)
```



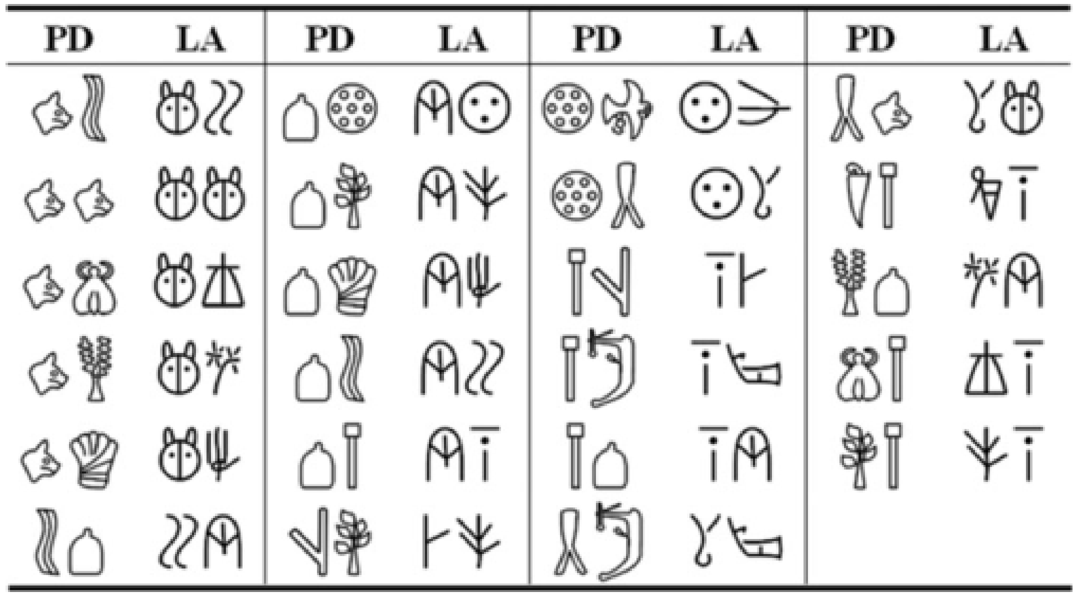
Now we see if we can get the same number of bigrams consisting of these syllabograms as Davis in the disc:

[253]:
```
# Use the provisional PD to LA mapping above to find common bigrams between LA␣
 ↪and the Disc
pd_inscription_as_la = list(map(lambda x: pd_la_davis_map[x] if x in␣
 ↪pd_la_davis_map else x, pd_inscription))
pd_inscription_as_la_words = ''.join(pd_inscription_as_la).split('|')
pd_la_bigrams = getNgrams(pd_inscription_as_la_words,2)

pd_bigrams_both = set([bg for bg in pd_la_bigrams if all(g in pd_la_davis_map.
 ↪values() for g in bg)])
#print(str(len(pd_bigrams_both)) + " bigrams", sorted(pd_bigrams_both))

pd_la_davis_map_r = {y:x for x,y in  pd_la_davis_map.items()}
df = pd.DataFrame([pd_bigrams_both,
                  [pd_la_davis_map_r[x[:1]] + pd_la_davis_map_r[x[-1:]] for x␣
 ↪in pd_bigrams_both]],
                  columns=[i+1 for i,p in enumerate(pd_bigrams_both)])
df = df.set_axis(['Linear A Bigrams', 'Disc Bigrams'], axis='index')
df.style.set_caption("Linear A and Phaistos Disc Bigrams").
 ↪set_table_styles(styles)
```



This matches the 23 bigrams given in Table 40 by Davis:

TABLE 40
Hypothetical LA homomorphs of the word-internal PD pairs in Table 34



Now we can count the number of pairs that also occur in the Linear A corpus.

[204]:
```
bg_both = sorted([(bg, la_bigrams.count(bg), pd_la_bigrams.count(bg))
                  for bg in pd_bigrams_both & set(la_bigrams)])

df = pd.DataFrame([[b for a,b,c in bg_both], [c for a,b,c in bg_both]],
                  columns=[a for a,b,c in bg_both])
df = df.set_axis(['Occurences in Linear A', 'Occurences in Disc'], axis='index')
df.style.set_table_styles(styles).set_caption("%i Bigrams that Appear in Both␣
↪Linear A and Phaistos Disc" % len(bg_both))
```

16 Bigrams that Appear in Both Linear A and Phaistos Disc

Occurences in Linear A	2	1	4	1	3	1	2	2	2	2	1	1	5	1	1	1
Occurences in Disc	2	1	1	3	2	1	1	1	2	1	6	3	1	2	2	1

We find only 16 instances of Disc bigrams appearing in Linear A. This is one less than found by Davis. Our output also gives the number of occurences of the bigrams in each of Linear A and the Disc, both as a total for all bigrams and for each bigram individually. So for ‘ ’ we find that it occurs twice in Linear A and once on the Phaistos Disc, i.e.: (‘ ’, 2, 1).

1.2 Reviewing the Results

Let’s take a look at bigram we are missing compared to Davis 2018:

```python
[7]: bg_pd_only = pd_bigrams_both - set(la_bigrams)
bg_pd_only = sorted([(bg, pd_la_bigrams.count(bg))
                     for bg in bg_pd_only])

df = pd.DataFrame([[b for a,b in bg_pd_only]],
                  columns=[a for a,b in bg_pd_only])
df = df.set_axis(["Occurences"], axis='index')
df.style.set_table_styles(styles).set_caption("Mapped bigrams that don't appear␣
↪in Linear A")
```

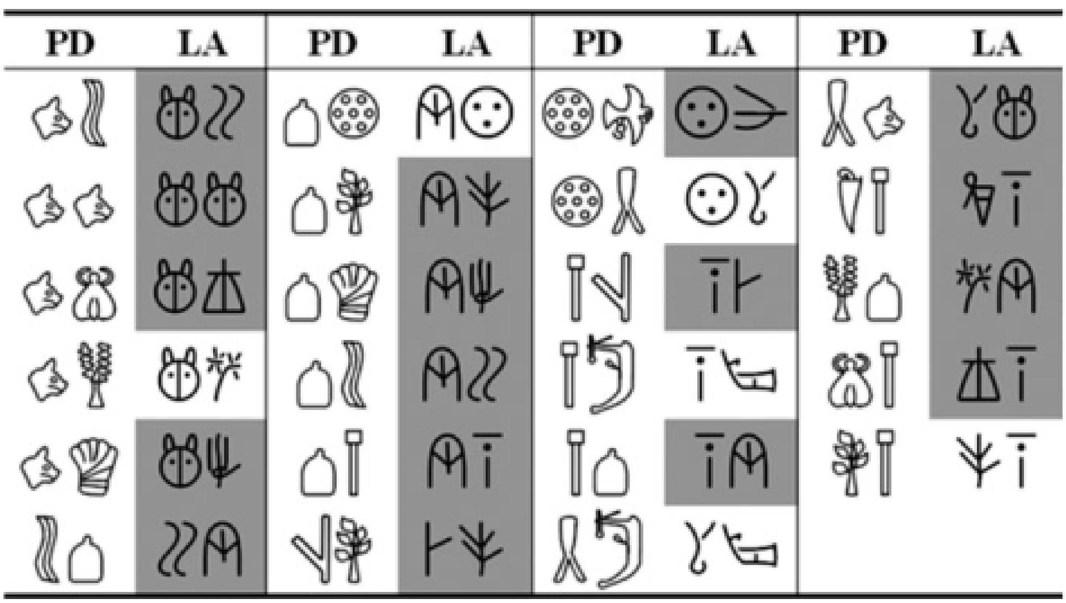
Mapped bigrams that don't appear in Linear A

Occurences	1	2	1	3	1	1	3

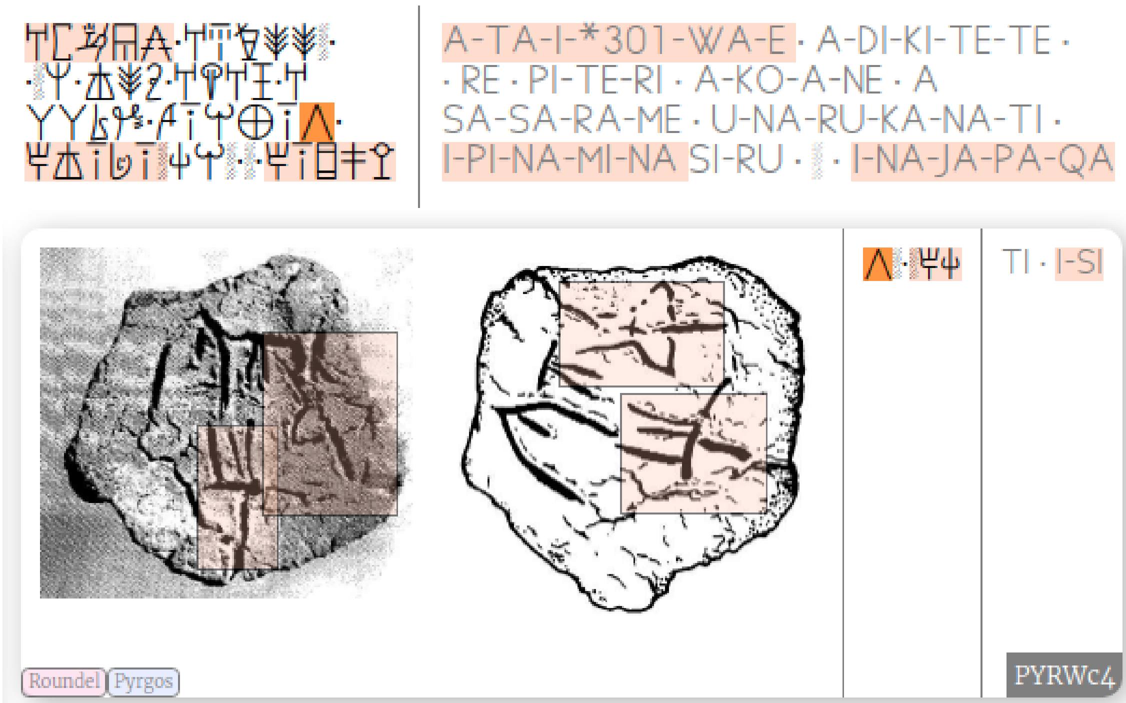
We can compare this with the table from (Davis 2018):

TABLE 42

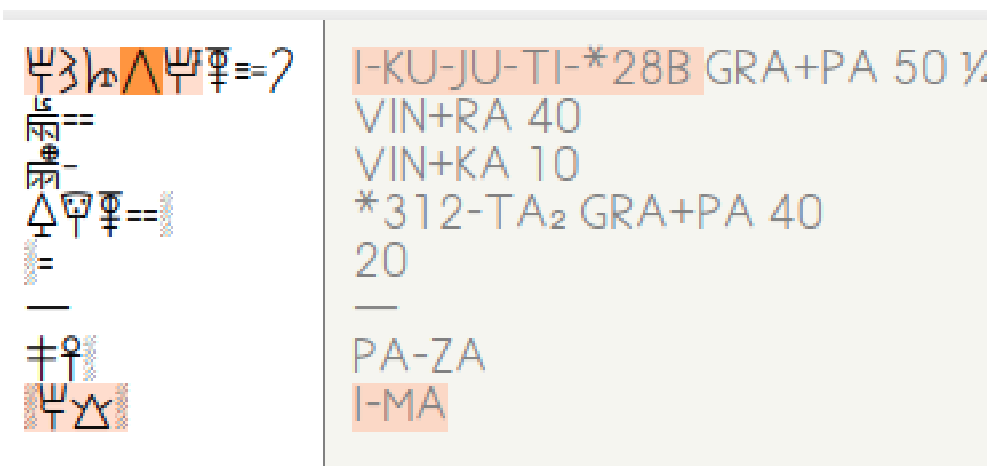
Reprise of Table 40; LA pairs also listed in Table 41 are highlighted



The difference is the bigram: ‘ ’, (when transliterated: TI-I). ‘ ’ (TI-I) does not actually appear in Linear A. Where the two syllabograms are adjacent they are not word-internal, i.e. they are in adjacent words rather than the same word:



Davis' probable source for the identification is a variation of (TI) which is *(28B). We find a single instance of ' ' (TI-*28B) in the Linear A corpus, in ZA6b:



It is not clear to me if it is valid to treat (TI-*28B) as the equivalent of (TI-I). If it is not, then the number of matching bi-grams between the Phaistos disc and the Linear A corpus must be revised down to 16. This no longer falls within the region of statistical significance, which Davis identifies as 16.4 or above.

1.3 Experimenting with Different Mappings

In this section we'll experiment with an expanded set of homomorphic mappings in the syllabograms of Linear A and the Phaistos disc and see if improves or changes the result of 17/23 observed by Davis.

[259]:
```
def runExperimentalMapping(exp_map):
    pd_inscription_as_la = list(map(lambda x: exp_map[x]
                                    if x in exp_map else x, pd_inscription))
    pd_inscription_as_la_words = ''.join(pd_inscription_as_la).split('|')
    pd_la_bigrams = getNgrams(pd_inscription_as_la_words,2)

    pd_bigrams_both = set([bg for bg in pd_la_bigrams
                           if all(g in exp_map.values() for g in bg)])


    bg_both = sorted([(bg, la_bigrams.count(bg), pd_la_bigrams.count(bg))
                      for bg in pd_bigrams_both & set(la_bigrams)])
    return (pd_bigrams_both, bg_both)

def displayExperimentalMappingResults(exp_both, exp_bigrams_both, exp_map):
    showDifferencesBetweenMappings(pd_la_davis_map, exp_map)

    exp_map_r = {y:x for x,y in exp_map.items()}
    df = pd.DataFrame([exp_bigrams_both,
                      [exp_map_r[x[:1]] + exp_map_r[x[-1:]] for x in␣
↪exp_bigrams_both]],
                      columns=[i+1 for i,p in enumerate(exp_bigrams_both)])

    df = df.set_axis(['Linear A Bigrams', 'Disc Bigrams'], axis='index')
    df = (df.style.set_caption("The %d Hypothetical Phaistos Disc Bigrams Along"
                                " With Their Hypothetical Linear A Counterparts"␣
↪% len(exp_bigrams_both))
            .set_table_styles(styles))
    display(df)

    df = pd.DataFrame([[b for a,b,c in exp_both] + [sum([b for a,b,c in␣
↪exp_both])],
                    [c for a,b,c in exp_both] + [sum([c for a,b,c in␣
↪exp_both])]],
                    columns=[a for a,b,c in exp_both] + ["Total"])
    df = df.set_axis(['Occurences in Linear A', 'Occurences in Disc'],␣
↪axis='index')
    df = (df.style.set_caption("The %d bigrams that actually appear in Linear␣
↪A" % len(exp_both))
            .set_table_styles(styles))
    display(df)

    bg_pd_only = exp_bigrams_both - set(la_bigrams)
    bg_pd_only = sorted([(bg, pd_la_bigrams.count(bg))
                      for bg in bg_pd_only])

    df = pd.DataFrame([[b for a,b in bg_pd_only]],
```

```
                  columns=[a for a,b in bg_pd_only])
    df = df.set_axis(["Occurences in the Phaistos Disc"], axis='index')
    df = (df.style.set_caption("The %d bigrams that don't appear in Linear A"
                               % (len(exp_bigrams_both) - len(bg_both)))
            .set_table_styles(styles))
    display(df)

def showDifferencesBetweenMappings(map1, map2):
    row_index = set([k for k in map1] + [k for k in map2])
    row_index = [a for a in row_index
                 if a not in map2 or a not in map1 or
                           map2[a] != map1[a]]

    df = pd.DataFrame([[map1[a] if a in map1 else "None"
                       for a in row_index],
                       [map2[a] if a in map2 else "None"
                        for a in row_index]]
                      , columns=row_index)
    df = df.set_axis(["Davis", "Ours"], axis='index')
    df = df.style.set_caption("Differences between Davis and our mapping").
 ↪set_table_styles(styles)
    display(df)
```

In the first instance we'll expand our mapping to include some additional glyphs and alter some others. The differences are given in the table below.

[249]:
```
pd_la_hogan_map = {
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
```

```
" ": " ",
" ": " ",
}

df = pd.DataFrame([pd_la_hogan_map.keys()
                  , pd_la_hogan_map.values()
                  ])
df = df.set_axis(["Phaistos Disc", "Linear A"], axis='index')
df = df.style.set_caption("Hypothetical Revised Mapping of Linear A and PD␣
↪Symbols").set_table_styles(styles)
display(df)
```

Hypothetical Revised Mapping of Linear A and PD Symbols

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Phaistos Disc	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Linear A	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Let's try this mapping:

[211]:
```
pd_bigrams_both, bg_both = runExperimentalMapping(pd_la_hogan_map)
displayExperimentalMappingResults(bg_both, pd_bigrams_both, pd_la_hogan_map)
```

Differences between Davis and our mapping

	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Davis	None	None	None	None	None	None	[illegible]	None	None	None	[illegible]
Ours	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	None	[illegible]	[illegible]	[illegible]	[illegible]

The 37 Hypothetical Phaistos Disc Bigrams Along With Their Hypothetical Linear A Counterparts

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
Linear A Bigrams	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Disc Bigrams	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

The 20 bigrams that actually appear in Linear A

	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	Total
Occurences in Linear A	2	1	4	1	2	1	3	1	2	2	2	2	2	6	1	2	3	1	1	1	40
Occurences in Disc	2	1	1	1	13	3	2	1	1	1	1	2	1	1	6	1	3	2	2	1	46

The 17 bigrams that don't appear in Linear A

	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Occurences in the Phaistos Disc	1	2	0	0	1	0	0	0	3	0	0	0	0	0	0	0	0

We get 37 possible bigrams, of which 20 actually appear in Linear A. Of the 17 that do not appear in Linear A, only 4 occur in the Phaistos disc. A poor result. When we inspect the bigrams that don't appear in Linear A we can see that 5 syllabograms in particular don't produce any result at at all. If we remove these as a bad lot and rerun the analysis again we get a much better result:

[250]:
```
del pd_la_hogan_map[" "]
del pd_la_hogan_map[" "]
del pd_la_hogan_map[" "]
del pd_la_hogan_map[" "]
del pd_la_hogan_map[" "]

df = pd.DataFrame([pd_la_hogan_map.keys()
                  , pd_la_hogan_map.values()
                  ])
df = df.set_axis(["Phaistos Disc", "Linear A"], axis='index')
df = df.style.set_caption("Hypothetical Revised Mapping of Linear A and PD␣
↪Symbols").set_table_styles(styles)
display(df)
```

Hypothetical Revised Mapping of Linear A and PD Symbols

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Phaistos Disc	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Linear A	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

[251]:
```
pd_bigrams_both, bg_both = runExperimentalMapping(pd_la_hogan_map)
displayExperimentalMappingResults(bg_both, pd_bigrams_both, pd_la_hogan_map)
```

Differences between Davis and our mapping

	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Davis	[illegible]	None	None	None	[illegible]	None	None	[illegible]
Ours	None	[illegible]	[illegible]	[illegible]	None	[illegible]	[illegible]	[illegible]

The 22 Hypothetical Phaistos Disc Bigrams Along With Their Hypothetical Linear A Counterparts

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Linear A Bigrams	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Disc Bigrams	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

The 20 bigrams that actually appear in Linear A

	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	Total
Occurences in Linear A	2	1	4	1	2	1	3	1	2	2	2	2	2	6	1	2	3	1	1	1	40
Occurences in Disc	2	1	1	1	13	3	2	1	1	1	1	2	1	1	6	1	3	2	2	1	46

The 2 bigrams that don't appear in Linear A

	[illegible]	[illegible]
Occurences in the Phaistos Disc	1	3

Now we have 20 mappings found in Linear A of a possible 22. This suggests our proposed modification to the mapping is better than Davis'.

1.4 Calculate the Statistical Significance

1.4.1 Recreate the Statistical Significance Results

To confirm we have a valid, improved mapping on Davis 2018 we'll first recreate the statistical significance results from Davis' paper before applying the same method to our mapping. We do this by running a million different random permutations of Davis' 14 proposed homomorphs and chart the results in the same way as Davis (2018).

[238]:
```
import itertools as it
import numpy as np

def getStatSignificance(list1, list2):
    buckets = {}
    for c in range(0, 1000000):
        n_map = {k:v for k,v in zip(np.random.permutation(list2), np.random.
 ↪permutation(list1))}
        pd_bigrams_both, bg_both = runExperimentalMapping(n_map)
        l = len(bg_both)
        if l in buckets:
            buckets[l] = buckets[l] + 1
        else:
            buckets[l] = 1
        if c % 50000 > 0:
            continue
    return buckets
```

[ ]:
```
dlist1 = list(pd_la_davis_map.values())
dlist2 = list(pd_la_davis_map.keys())

davis_results = getStatSignificance(dlist1, dlist2)
print(davis_results)
```

[242]:
```
from numpy import mean


def printStatSigResult(results, score):
    table = sorted([
                  (k,
                   '{:,}'.format(v),
                   "{:.4%}".format(v / sum(results.values()))
                  )
                  for k,v in results.items()
              ], key=lambda x: x[0])
    df = pd.DataFrame(table,
```

```
                    columns=["Score out of 23", "Permutations with that score",␣
↪"% of Permutations"])
    df = df.style.hide_index().set_caption("Syllabotactic similarity scores␣
↪produced by 1,000,000 "
        "different random associations between the 14 PD and LA signs").
↪set_table_styles(styles)
    display(df)

    l = list(it.chain.from_iterable([[k] * v for k,v in results.items()]))
    sd = np.std(l)
    avg = mean(l)
    print("Average Score : " + "{:.4}".format(avg))
    print("Standard Deviation : " + "{:.4}".format(sd))
    print("Average Score + 2 standard deviations: " + "{:.4}".format(avg +␣
↪(sd*2)))

    p_val = sum([
                (v / sum(results.values()))
                for k,v in results.items()
                if k > (score - 1)
                ])

    print("P Value " + "{:.4}".format(p_val))

printStatSigResult(davis_results, 17)
```

Syllabotactic similarity scores produced by 1,000,000 different random associations between the 14 PD and LA signs

Score out of 23	Permutations with that score	% of Permutations
1	34	0.0034%
2	216	0.0216%
3	1,047	0.1047%
4	4,260	0.4260%
5	12,007	1.2007%
6	27,897	2.7897%
7	53,641	5.3641%
8	87,713	8.7713%
9	123,224	12.3224%
10	148,836	14.8836%
11	157,938	15.7938%
12	142,513	14.2513%
13	109,039	10.9039%
14	70,580	7.0580%
15	37,675	3.7675%
16	16,129	1.6129%
17	5,466	0.5466%
18	1,485	0.1485%
19	266	0.0266%
20	31	0.0031%
21	3	0.0003%

```
Average Score : 10.72
Standard Deviation : 2.48
```

```
Average Score + 2 standard deviations: 15.68
P Value 0.007251
```

We compare this result with Davis' findings and they are similar, in fact they are slightly better:

TABLE 52
Syllabotactic similarity scores produced by 1,000,000 different random associations between the 14 PD and LA signs in Table 48

Score out of 23:	**Permutations with that score:**	**% of permutations:**
0	1	0.0001%
1	15	0.0015%
2	128	0.0128%
3	714	0.0714%
4	2739	0.2739%
5	8070	0.8070%
6	19697	1.9697%
7	38747	3.8747%
8	67967	6.7967%
9	100032	10.0032%
10	130698	13.0698%
11	149038	14.9038%
12	149347	14.9347%
13	129120	12.9120%
14	96812	9.6812%
15	60141	6.0141%
16	30145	3.0145%
17	11910	1.1910%
18	3719	0.3719%
19	822	0.0822%
20	119	0.0119%
21	18	0.0018%
22	1	0.0001%
Total permutations:	1,000,000	100%
Average score out of 23:	11.332	
Standard deviation (σ):	2.560	
Average score + 2σ:	16.453	
Score of 17/23:	Average + 2.2σ	p = 0.0166

Now we apply the same procedure to our own mapping.

[239]:
```python
hlist1 = list(pd_la_hogan_map.values())
hlist2 = list(pd_la_hogan_map.keys())

hogan_results = getStatSignificance(hlist1, hlist2)
```

[245]:
```python
printStatSigResult(hogan_results, 20)
```

Syllabotactic similarity scores produced by 1,000,000 different random associations between the 14 PD and LA signs

Score out of 23	Permutations with that score	% of Permutations
2	14	0.0014%
3	65	0.0065%
4	529	0.0529%
5	2,221	0.2221%
6	7,116	0.7116%
7	18,221	1.8221%
8	39,368	3.9368%
9	69,854	6.9854%
10	106,190	10.6190%
11	139,836	13.9836%
12	159,143	15.9143%
13	155,589	15.5589%
14	129,496	12.9496%
15	89,745	8.9745%
16	50,189	5.0189%
17	22,326	2.2326%
18	7,824	0.7824%
19	1,929	0.1929%
20	310	0.0310%
21	33	0.0033%
22	2	0.0002%

```
Average Score : 12.19
Standard Deviation : 2.427
Average Score + 2 standard deviations: 17.04
P Value 0.000345
```

Our p-value is much lower, suggesting that we have a superior mapping to that proposed by Davis (2018).

1.5 Comparing Word-End Syllabograms

Let's compare glyphs that appear at the end of words in Linear A and the Disc.

[29]:
```
syllables = {
' ': 'DA', ' ': 'RO', ' ': 'PA', ' ': 'TE', ' ': 'TO', ' ': 'NA',
' ': 'DI', ' ': 'A', ' ': 'SE', ' ': 'U', ' ': 'PO', ' ': 'ME',
' ': 'QA', ' ': 'ZA', ' ': 'ZO', ' ': 'QI', ' ': 'MU', ' ': 'NE',
' ': 'RU', ' ': 'RE', ' ': 'I', ' ': 'PU ', ' ': 'NI', ' ': 'SA',
' ': 'TI', ' ': 'E', ' ': 'PI', ' ': 'WI', ' ': 'SI', ' ': 'KE',
' ': 'DE', ' ': 'JE', ' ': 'NWA', ' ': 'PU', ' ': 'DU', ' ': 'RI',
' ': 'WA', ' ': 'NU', ' ': 'PA ', ' ': 'JA', ' ': 'SU', ' ': 'TA',
' ': 'RA', ' ': 'O', ' ': 'JU', ' ': 'TA ', ' ': 'KI', ' ': 'TU',
' ': 'KO', ' ': 'MI', ' ': 'ZE', ' ': 'RA ', ' ': 'KA', ' ': 'QE',
' ': 'MA', ' ': 'KU', ' ': 'AU', ' ': 'TWE', ' ': 'ZU'
}

vowels = {
' ': 'A',
' ': 'E',
' ': 'I',
```

```
' ': 'O',
' ': 'U',
' ': 'AU',
}
```

Let's find the most common last syllabograms in Linear A words:

[395]:
```python
import collections

la_last_letters = { l[-1:]: len([w for w in la_words if w[-1:] == l[-1:]])
                    for l in la_words if len(l) > 1 and l[-1:] in syllables}
# Sort highest to top
la_last_letters = sorted(la_last_letters.items(), key=lambda x:x[1],␣
↪reverse=True)
la_last_letters = collections.OrderedDict(la_last_letters)

r = {key: rank for rank, key in enumerate(sorted(set(la_last_letters.values()),␣
↪reverse=True), 1)}
la_last_letters_ranked = {k: r[v] for k,v in la_last_letters.items()}

df = pd.DataFrame([[b for a,b in la_last_letters.items()],
                   [b for a,b in la_last_letters_ranked.items()]],
                columns=[a for a,b in la_last_letters.items()])
df = df.set_axis(['Occurrences', 'Ranking'], axis='index')
df = df.style.set_caption("Most Common Word-End Syllabograms in Linear A").
↪set_table_styles(styles)
display(df)
```

Most Common Word-End Syllabograms in Linear A

Occurrences	48	45	44	43	41	37	32	28	27	27	23	22	21	20	20	19	18	17
Ranking	1	2	3	4	5	6	7	8	9	9	10	11	12	13	13	14	15	16

Occurrences	17	16	16	16	15	15	14	14	14	14	13	13	13	12	12	11	10	9
Ranking	16	17	17	17	18	18	19	19	19	19	20	20	20	21	21	22	23	24

Occurrences	9	8	8	8	8	7	7	6	6	6	5	5	4	3	2	2	1
Ranking	24	25	25	25	25	26	26	27	27	27	28	28	29	30	31	31	32

And do the same for the disc:

[542]:
```python
pd_last_letters = { l[-1:]: len([w for w in pd_words if w[-1:] == l[-1:]])
                    for l in pd_words if len(l) > 1}
# Sort highest to top
pd_last_letters = sorted(pd_last_letters.items(), key=lambda x:x[1],␣
↪reverse=True)
pd_last_letters = collections.OrderedDict(pd_last_letters)

r = {key: rank for rank, key in enumerate(sorted(set(pd_last_letters.values()),␣
↪reverse=True), 1)}
pd_last_letters_ranked = {k: r[v] for k,v in pd_last_letters.items()}

df = pd.DataFrame([[b for a,b in pd_last_letters.items()],
                   [b for a,b in pd_last_letters_ranked.items()]],
```

```
                 columns=[a for a,b in pd_last_letters.items()])
df = df.set_axis(['Occurrences', 'Ranking'], axis='index')
df = df.style.set_caption("Most Common Word-End Syllabograms in PD (By␣
↪Occurrence)").set_table_styles(styles)
display(df)
```

Most Common Word-End Syllabograms in PD (By Occurrence)

Occurrences	8	7	7	6	4	4	3	3	3	3	2	2	1	1	1	1	1	1	1	1	1
Ranking	1	2	2	3	4	4	5	5	5	5	6	6	7	7	7	7	7	7	7	7	7

[529]:
```
pd_la_full_map = {
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
" ": " ",
}

"""
ranking_comp = sorted(
    [[k, pd_last_letters_ranked[k], pd_la_full_map[k],␣
↪la_last_letters_ranked[pd_la_full_map[k]]]
    for k,v in pd_la_full_map.items() if k in pd_last_letters and v in␣
↪la_last_letters]
    , key=lambda x: abs(x[1] - x[3]))

df = pd.DataFrame(ranking_comp,
                 columns=["PD Glyph", "PD Ranking", "LA Glyph", "LA Ranking"])
df = df.style.hide_index().set_caption("Raw Ranking").set_table_styles(styles)
display(df)
```

```
"""

n_ranking_comp = sorted([
                     (k,
                      pd_last_letters_ranked[k],
                      pd_la_full_map[k],
                      max(1, int((la_last_letters_ranked[pd_la_full_map[k]]
                                  / len(la_last_letters_ranked))
                                 * max([c for b,c in pd_last_letters_ranked.
 ↪items()])))
                     )
                     for k,v in pd_la_full_map.items() if k in pd_last_letters␣
 ↪and v in la_last_letters
                 ], key=lambda x: abs(x[1] - x[3]))

df = pd.DataFrame(n_ranking_comp,
                columns=["PD Glyph", "PD Ranking", "LA Glyph", "LA Ranking"])
df = df.style.hide_index().set_caption("Normalized Ranking").
 ↪set_table_styles(styles)
display(df)
```

Normalized Ranking

PD Glyph	PD Ranking	LA Glyph	LA Ranking
[illegible]	1	[illegible]	1
[illegible]	2	[illegible]	1
[illegible]	4	[illegible]	1
[illegible]	7	[illegible]	3
[illegible]	7	[illegible]	2
[illegible]	7	[illegible]	2
[illegible]	6	[illegible]	1
[illegible]	7	[illegible]	1

1.6 Compare Word-Initial Syllabograms

Let's find the most common last syllabograms in Linear A words:

[543]:
```
import collections

la_first_letters = { l[:1]: len([w for w in la_words if w[:1] == l[:1]])
                     for l in la_words if len(l) > 1 and l[:1] in syllables}
# Sort highest to top
la_first_letters = sorted(la_first_letters.items(), key=lambda x:x[1],␣
 ↪reverse=True)
la_first_letters = collections.OrderedDict(la_first_letters)

r = {key: rank for rank, key in enumerate(sorted(set(la_first_letters.
 ↪values()), reverse=True), 1)}
la_first_letters_ranked = {k: r[v] for k,v in la_first_letters.items()}

df = pd.DataFrame([[b for a,b in la_first_letters.items()],
```

```
                        [b for a,b in la_first_letters_ranked.items()]],
                    columns=[a for a,b in la_first_letters.items()])
df = df.set_axis(['Occurrences', 'Ranking'], axis='index')
df = df.style.set_caption("Most Common Word-Initial Syllabograms in Linear A␣
↪(By Occurrence)").set_table_styles(styles)
display(df)
```

Most Common Word-Initial Syllabograms in Linear A (By Occurrence)

Occurrences	115	51	51	36	35	33	29	29	28	26	25	24	24	23	22	20	18	18	17	15	14	14	14	12	12	11	11	10	9	9	8	8	8	7	7	7	7	6	6	6	6	4	3	3	3	3	2	2	2	2	2	2	2	2	1
Ranking	1	2	2	3	4	5	6	6	7	8	9	10	10	11	12	13	14	14	15	16	17	17	17	18	18	19	19	20	21	21	22	22	22	23	23	23	23	24	24	24	24	25	26	26	26	26	27	27	27	27	27	27	27	27	28

And do the same for the disc:

[545]:
```
pd_first_letters = { l[:1]: len([w for w in pd_words if w[:1] == l[:1]])
                        for l in pd_words if len(l) > 1}
# Sort highest to top
pd_first_letters = sorted(pd_first_letters.items(), key=lambda x:x[1],␣
↪reverse=True)
pd_first_letters = collections.OrderedDict(pd_first_letters)

r = {key: rank for rank, key in enumerate(sorted(set(pd_first_letters.
↪values()), reverse=True), 1)}
pd_first_letters_ranked = {k: r[v] for k,v in pd_first_letters.items()}

df = pd.DataFrame([[b for a,b in pd_first_letters.items()],
                        [b for a,b in pd_first_letters_ranked.items()]],
                    columns=[a for a,b in pd_first_letters.items()])
df = df.set_axis(['Occurrences', 'Ranking'], axis='index')
df = df.style.set_caption("Most Common Word-Initial Syllabograms in PD (By␣
↪Occurrence)").set_table_styles(styles)
display(df)
```

Most Common Word-Initial Syllabograms in PD (By Occurrence)

Occurrences	19	8	5	4	4	2	2	2	2	2	2	2	1	1	1	1	1	1	1
Ranking	1	2	3	4	4	5	5	5	5	5	5	5	6	6	6	6	6	6	6

[532]:
```
"""
ranking_comp = [(k, pd_first_letters_ranked[k], pd_la_full_map[k],␣
↪la_first_letters_ranked[pd_la_full_map[k]])
                    for k,v in pd_la_full_map.items() if k in pd_first_letters and␣
↪v in la_first_letters]

df = pd.DataFrame(ranking_comp,
                    columns=["PD Glyph", "PD Ranking", "LA Glyph", "LA Ranking"])
df = df.style.hide_index().set_caption("Raw Ranking").set_table_styles(styles)
display(df)
```

```
"""

n_ranking_comp = sorted([
                    (k,
                     pd_first_letters_ranked[k],
                     pd_la_full_map[k],
                     max(1, int((la_first_letters_ranked[pd_la_full_map[k]]
                                 / len(la_first_letters_ranked))
                                * max([c for b,c in pd_first_letters_ranked.
 ↪items()])))
                    )
                    for k,v in pd_la_full_map.items() if k in pd_first_letters␣
 ↪and v in la_first_letters
                ], key=lambda x: abs(x[1] - x[3]))

df = pd.DataFrame(n_ranking_comp,
                columns=["PD Glyph", "PD Ranking", "LA Glyph", "LA Ranking"])
df = df.style.hide_index().set_caption("Normalized Ranking").
 ↪set_table_styles(styles)
display(df)
```

Normalized Ranking

PD Glyph	PD Ranking	LA Glyph	LA Ranking
[illegible]	1	[illegible]	1
[illegible]	2	[illegible]	1
[illegible]	4	[illegible]	1
[illegible]	4	[illegible]	1
[illegible]	6	[illegible]	2
[illegible]	5	[illegible]	1
[illegible]	6	[illegible]	2
[illegible]	5	[illegible]	1
[illegible]	6	[illegible]	2
[illegible]	6	[illegible]	1

1.7 Examining Potential Illegal Combinations

[19]:
```
syllables = {
' ': 'DA', ' ': 'RO', ' ': 'PA', ' ': 'TE', ' ': 'TO', ' ': 'NA',
' ': 'DI', ' ': 'A', ' ': 'SE', ' ': 'U', ' ': 'PO', ' ': 'ME',
' ': 'QA', ' ': 'ZA', ' ': 'ZO', ' ': 'QI', ' ': 'MU', ' ': 'NE',
' ': 'RU', ' ': 'RE', ' ': 'I', ' ': 'PU ', ' ': 'NI', ' ': 'SA',
' ': 'TI', ' ': 'E', ' ': 'PI', ' ': 'WI', ' ': 'SI', ' ': 'KE',
' ': 'DE', ' ': 'JE', ' ': 'NWA', ' ': 'PU', ' ': 'DU', ' ': 'RI',
' ': 'WA', ' ': 'NU', ' ': 'PA ', ' ': 'JA', ' ': 'SU', ' ': 'TA',
' ': 'RA', ' ': 'O', ' ': 'JU', ' ': 'TA ', ' ': 'KI', ' ': 'TU',
' ': 'KO', ' ': 'MI', ' ': 'ZE', ' ': 'RA ', ' ': 'KA', ' ': 'QE',
' ': 'MA', ' ': 'KU', ' ': 'AU', ' ': 'TWE', ' ': 'ZU'
}

vowels = {
' ': 'A',
```

```
' ': 'E',
' ': 'I',
' ': 'O',
' ': 'U',
' ': 'AU',
}
```

[20]:
```
v_bgs = [(syllables[l[:1]][1:2] if l[:1] in syllables else "?")
             + (syllables[l[1:2]] if l[1:2] in syllables else "?")
        for l in la_bigrams if l[1:2] in vowels]
legal_vowel_combos_la = sorted(set([(bg, v_bgs.count(bg)) for bg in v_bgs]),␣
↪key=lambda x:x[1], reverse=True)

df = pd.DataFrame([[b for a,b in legal_vowel_combos_la]],
                columns=[a for a,b in legal_vowel_combos_la])
df = df.set_axis(['Ranking'], axis='index')
df = df.style.set_caption("Adjoining Vowels in Linear A").
↪set_table_styles(styles)
display(df)
```

Adjoining Vowels in Linear A

	AI	AU	AA	EI	?I	OA	EA	IA	IE	AO	AE	?U	OI	?A	?E	UU	UA	II	I	UE	OE	AAU	EU	IO	OAU	UI
Ranking	33	12	8	7	5	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	1	1	1	1	1

[21]:
```
v_bgs = [(syllables[l[:1]][1:2] if l[:1] in syllables else "?")
             + (syllables[l[1:2]] if l[1:2] in syllables else "?")
        for l in pd_bigrams_both if l[1:2] in vowels]

legal_vowel_combos_pd = sorted(set([(bg, v_bgs.count(bg)) for bg in v_bgs]),␣
↪key=lambda x:x[1], reverse=True)

df = pd.DataFrame([[b for a,b in legal_vowel_combos_pd]],
                columns=[a for a,b in legal_vowel_combos_pd])
df = df.set_axis(['Ranking'], axis='index')
df = df.style.set_caption("Adjoining Vowels in PD").set_table_styles(styles)
display(df)
```

Adjoining Vowels in PD

Ranking

[583]:
```
possible_vowel_combos = [a+b for a,b in list(it.product(vowels.values(),vowels.
↪values()))]
vowel_combos_in_la = [a for a,b in legal_vowel_combos_la]
vowel_combos_not_in_la = [a for a in possible_vowel_combos if a not in␣
↪vowel_combos_in_la]
```