AN ATTEMPT TO DECRYPT PAGES 269–271 OF JONATHAN SAFRAN FOER'S *Extremely Loud & Incredibly Close*

A PREPRINT

Yannis Haralambous
Département Informatique
IMT Atlantique
CS 83818
29238 Brest Cedex 3
yannis.haralambous@imt-atlantique.fr

November 16, 2020

ABSTRACT

In this paper we attempt to decrypt the sequence of digits given by Jonathan Safran Foer in his novel *Extremely Loud & Incredibly Close*. We create directed acyclic graphs that a human can follow to find potential solutions. Representations of these graphs are displayed in this paper. The Python code used to produce them is also provided, in the appendix.

Keywords Foer, extremely loud, digits, decrypting.

This paper contains supplementary material to [1]. The PDF files of graphs described in this paper, at full size, can be downloaded here: `https://www.imt-atlantique.fr/sites/default/files/users/user1063/foer-graphs.pdf` and here: `https://www.imt-atlantique.fr/sites/default/files/users/user1063/foer-graphs2.pdf`

1 Introduction

In Foer's *Extremely Loud & Incredibly Close* [2, p. 269–271], Oskar's grand-father calls his grand-mother on the phone, but, being mute, has no other solution than using the ISO/IEC 9995-8 [3] mapping of letters on telephone keypads to match letters of his message into digits, and then pushing the corresponding keys so that the dual-tone multi-frequency signals are played. Foer provides the digits (the sounds of chiwh would have been perceived by Oskar's grand-mother, assuming she has the perfect pitch):

> 6, 9, 6, 2, 6, 3, 4, 7, 3, 5, 4, 3, 2, 5, 8, 6, 2, 6, 3, 4, 5, 8, 7, 8, 2, 7, 7, 4, 8, 3, 3, 2, 8, 8, 4, 3, 2, 4, 7, 7,
> 6, 7, 8, 4, 6, 3, 3, 3, 8, 6, 3, 4, 6, 3, 6, 7, 3, 4, 6, 5, 3, 5, 7! 6, 4, 3, 2, 2, 6, 7, 4, 2, 5, 6, 3, 8, 7, 2, 6, 3,
> 4, 3? 5, 7, 6, 3, 5, 8, 6, 2, 6, 3, 4, 5, 8, 7, 8, 2, 7, 7, 4, 8, 3, 9, 2, 8, 8, 4, 3, 2, 4, 7, 7, 6, 7, 8, 4, 6, 3, 3,
> 3, 8! 4, 3, 2, 4, 7, 7, 6, 7, 8, 4! 6, 3, 3, 3, 8, 6, 3, 9, 6, 3, 6, 6, 3, 4, 6, 5, 3, 5, 7! 6, 4, 3, 2, 2, 6, 7, 4, 2,
> 5, 6, 3, 8, 7, 2, 6, 3, 4, 3? 5, 7, 6, 3, 5, 8, 6, 2, 6, 3, 4, 5, 8, 7, 8, 2, 7, 7, 4, 8, 3, 3, 2, 8! 7, 7, 4, 8, 3, 3,
> 2, 8, 3, 4, 3, 2, 4, 7, 6, 6, 7, 8, 4, 6, 8, 3, 8, 8, 6, 3, 4, 6, 3, 6, 7, 3, 4, 6, 7, 7, 4, 8, 3, 3, 9, 8, 8, 4, 3, 2,
> 4, 5, 7, 6, 7, 8, 4, 6, 3, 5, 5, 2, 6, 9, 4, 6, 5, 6, 7, 5, 4, 6! (... another 2,317 digits)

together with punctuation marks indicating sentence boundaries. The text contains 129 sentences, 92 of which end with an exclamation mark and the rest by a question mark. (It is not clear how the punctuation marks are transmitted through the phone.) Mapping letters to digits according to ISO/IEC 9995-8 is a lossy operation since to each digit correspond 3 or 4 letters. On page 269, Foer gives some examples that can be elucidated without much effort (our solutions given in brackets):

> I pressed 4, 3, 5,ă5, 6," [HELLO] she said, Hello?" I asked, 4, 7, 4, 8, 7, 3, 2, 5, 5, 9, 9, 6, 8?"
> [ISITREALLYYOU] She said, Your phone isn't one hundred dollars. Hello?" I wanted to reach my

hand through the mouthpiece, down the line, and into her room, I wanted to reach YES, I asked, 4, 7, 4, 8, 7, 3, 2, 5, 5, 9, 9, 6, 8?" [ISITREALLYYOU] She said, Hello?" I told her, 4, 3, 5, 7! [HELP]

2 The Corpus

Here is the complete set of sentences and terminating punctuation marks, numbered in increasing order:

1. 69626347354325862634587827748332884324776784633386346367346535 7!
2. 6432267425638726343?
3. 576358626345878277483928843247767846338634636734653 57!
4. 4324776784!
5. 6333863963663465357!
6. 6432267425638726343?
7. 57635862634587827748332 8!
8. 774833283432476678468388634636734677483398843245767846355269465675 46!
9. 526265952?
10. 69626547554525264624527227742552924526!
11. 42265425745252626545272277425522245 2!
12. 72277425522245247227246555265465675 4!
13. 4324336384!
14. 6333863963663465353!
15. 2233263425638326343?
16. 5683?
17. 53635862634583823483328!
18. 33483328343247667846838863463!
19. 227742552924526!
20. 42265425745252626545272277425522245 2!
21. 72277425522245247227246555265465675 4!
22. 65557!
23. 64522674256526!
24. 26545?
25. 57655262654527227742592224524556524655 52!
26. 45245565!
27. 5683?
28. 55655262634583823348392884324336384633 38!
29. 432433638463!
30. 5683?
31. 5683?
32. 5683!
33. 42265425745252626545272274524635862634587827748332 8!
34. 65557!
35. 64522674256526!
36. 26545?
37. 57655262654527227742592224524!
38. 5683?

39. 5565246367346774833988432457678463552694656754 6!
40. 526265952?
41. 6962654755452526462452722774255292452 6!
42. 4226542574525262654527227742552224 52!
43. 7227742552224524722724655526546567 54!
44. 65557!
45. 64522674256526!
46. 26545?
47. 5765526265452722774259222 4524!
48. 5683?
49. 55652465552!
50. 45245565!
51. 2552924526!
52. 4226542!
53. 5565526263458382334839288432433638463338!
54. 4324336384!
55. 6333863963663465353!
56. 2233263425638326343?
57. 5683?
58. 536358626345838233483328!
59. 272465552654656754!
60. 65557!
61. 64522674256526!
62. 26545?
63. 576552626545272277425922245245565246555 2!
64. 45245565!
65. 5683?
66. 55655262634583823348392884324346555 2!
67. 45245565!
68. 6545?
69. 45?
70. 5565526263458382334839288432433638463338!
71. 4324336384!
72. 633367425638726343?
73. 576358626345878277483328!
74. 774833283432476678468388634636734677483398843245767846355269465675 46!
75. 526265952?
76. 6962654755452526462452722774255292452 6!
77. 4226542574525262654527227742552224 52!
78. 7227742552224524722724655526546567 54!
79. 65557!
80. 64522674256526!
81. 26545?
82. 5765526265452722774259222 4524!

83. 5683?
84. 55652465552!
85. 45245565!
86. 8639636634653532233263425638326343?
87. 5683?
88. 536358626345838233483328!
89. 33483328343247667846838863463!
90. 2277467425638726343?
91. 57635862634587827748332 8!
92. 774833283432476678468388634636734677483398843245767846355269465675 46!
93. 526265952?
94. 6962654755452526462452722774255292 4526!
95. 42265425745252626545272277425522245 2!
96. 72277425522245247227246555265465675 4!
97. 65557!
98. 64522674256526!
99. 26545?
100. 576552626545272277425922245 24!
101. 5683?
102. 55652465552!
103. 45245565!
104. 2552924526!
105. 4226542!
106. 556552626345838233483928843243363846333 8!
107. 4324336384!
108. 6333863963663465353!
109. 2233263425638326343?
110. 5683?
111. 536358626345838233483328!
112. 272465552654656754!
113. 65557!
114. 64522674256526!
115. 26545?
116. 576552626545272277425922245245565246555 2!
117. 45245565!
118. 5683?
119. 556552626345838233483928843243363846333 8!
120. 432433638463!
121. 5683?
122. 5683?
123. 5683!
124. 42265425745252626545272274524635862634587827748332 8!
125. 774833283432476678468388634636734677483398843245767846355269465675 46!
126. 526265952?

127. 69626545652465552742552224 52!
128. 72277425522245247227246555265465675 4!
129. 65557!

3 Sentence Repetition

When inspecting the 129 sentences one realizes that most of them are in fact repetitions of previous sentences. The correspondences are as follows:

Id	Status	Class	Punct.
1	new	1	!
2	new	2	?
3	new	3	!
4	new	4	!
5	new	5	!
6	old	2	?
7	new	6	!
8	new	7	!
9	new	8	?
10	new	9	!
11	new	10	!
12	new	11	!
13	new	12	!
14	new	13	!
15	new	14	?
16	new	15	?
17	new	16	!
18	new	17	!
19	new	18	!
20	old	10	!
21	old	11	!
22	new	19	!
23	new	20	!
24	new	21	?
25	new	22	!
26	new	23	!
27	old	15	?
28	new	24	!
29	new	25	!
30	old	15	?
31	old	15	?
32	old	15	!
33	new	26	!
34	old	19	!
35	old	20	!
36	old	21	?
37	new	27	!
38	old	15	?
39	new	28	!
40	old	8	?
41	old	9	!
42	old	10	!
43	old	11	!

Id	Status	Class	Punct.
44	old	19	!
45	old	20	!
46	old	21	?
47	old	27	!
48	old	15	?
49	new	29	!
50	old	23	!
51	new	30	!
52	new	31	!
53	old	24	!
54	old	12	!
55	old	13	!
56	old	14	?
57	old	15	?
58	new	32	!
59	new	33	!
60	old	19	!
61	old	20	!
62	old	21	?
63	old	22	!
64	old	23	!
65	old	15	?
66	new	34	!
67	old	23	!
68	new	35	?
69	new	36	?
70	old	24	!
71	old	12	!
72	new	37	?
73	old	6	!
74	old	7	!
75	old	8	?
76	old	9	!
77	old	10	!
78	old	11	!
79	old	19	!
80	old	20	!
81	old	21	?
82	old	27	!
83	old	15	?
84	old	29	!
85	old	23	!
86	new	38	?

Id	Status	Class	Punct.
87	old	15	?
88	old	32	!
89	old	17	!
90	new	39	?
91	old	6	!
92	old	7	!
93	old	8	?
94	old	9	!
95	old	10	!
96	old	11	!
97	old	19	!
98	old	20	!
99	old	21	?
100	old	27	!
101	old	15	?
102	old	29	!
103	old	23	!
104	old	30	!
105	old	31	!
106	old	24	!
107	old	12	!
108	old	13	!
109	old	14	?
110	old	15	?
111	old	32	!
112	old	33	!
113	old	19	!
114	old	20	!
115	old	21	?
116	old	22	!
117	old	23	!
118	old	15	?
119	old	24	!
120	old	25	!
121	old	15	?
122	old	15	?
123	old	15	!
124	old	26	!
125	old	7	!
126	old	8	?
127	new	40	!
128	old	11	!
129	old	19	!

It is noticeable that all sentence repetitions share the same punctuation mark, with a single exception: sentence class 15 has 17 instances, 15 of which (16, 27, 30, 31, 38, 48, 57, 65, 83, 87, 101, 110, 118, 121, 122) are terminated by a question mark and 2 (32 and 123) by an exclamation mark. This sentence is very particular since it is very short (4 digits: 5683), there are two obvious solutions: LOVE and LOUD.

4 Our Method

In order to compare numeric sequences with words, we needed a corpus. We used the following:

1. the complete set of words of the novel [2]: it consists of 8,518 sentences containing 87,590 word instances belonging to 5,660 distinct words;

2. the complete set of unmunched words of the US English dictionary of the Hunspell spelling checker: 104,682 distinct words.

We have removed all non-ASCII letter characters and converted everything to uppercase.

The Python code provided in the Appendix produces 39 PDF files representing the sentence classes (sentence 69 = class 36 is too short to be processed).

5 How to Find Candidate Sentences

To find candidate sentences one needs to follow arrows from the left of the graph to its right. Every column corresponds to a given offset in the number sequence, and different colors are attached to columns. Arrows leaving a node use the color of their destination.

Let us take an example: let us take sentence 1 (file `sentence1.pdf`, on page 6 of this document). On the extreme left we have a column of words connected by red arrows, they are the candidates for the initial word of the sentence. We recognize OWN, MY and NY (New York). Arrows leaving OWN are yellow so we need to move three horizontal units ahead (since OWN has three letters) to the yellow column containing words AND, AMD, etc., obviously a dead-end. Starting with MY we have to move two units ahead, and this why arrows leaving MY are blue. On the blue column we find NAME, OBOE and MAN. We continue moving until the end.

There is a problem though: our list of words, however rich it is, does not contain all possible proper names. In this example we have proper names ELIE ALTO (or ELI DALTO) and MR (or MS) FINKEL but only the FINK part has been detected by the algorithm. This makes search more difficult and proper names have to be added manually, we haven't found any better solution for the moment.

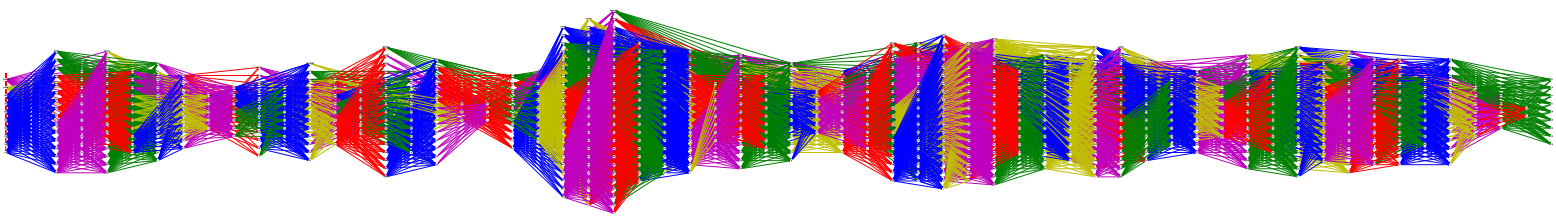
References

[1] Yannis Haralambous, Frédéric Landragin, and Kenichi Handa. Graphemic and graphetic methods in speculative fiction. In Yannis Haralambous, editor, *Proccedings of the Grapholinguistics in the 21st Century 2020 Conference*, Brest, 2021. Fluxus Editions. `http://www.fluxus-editions.fr/gla4.php`.

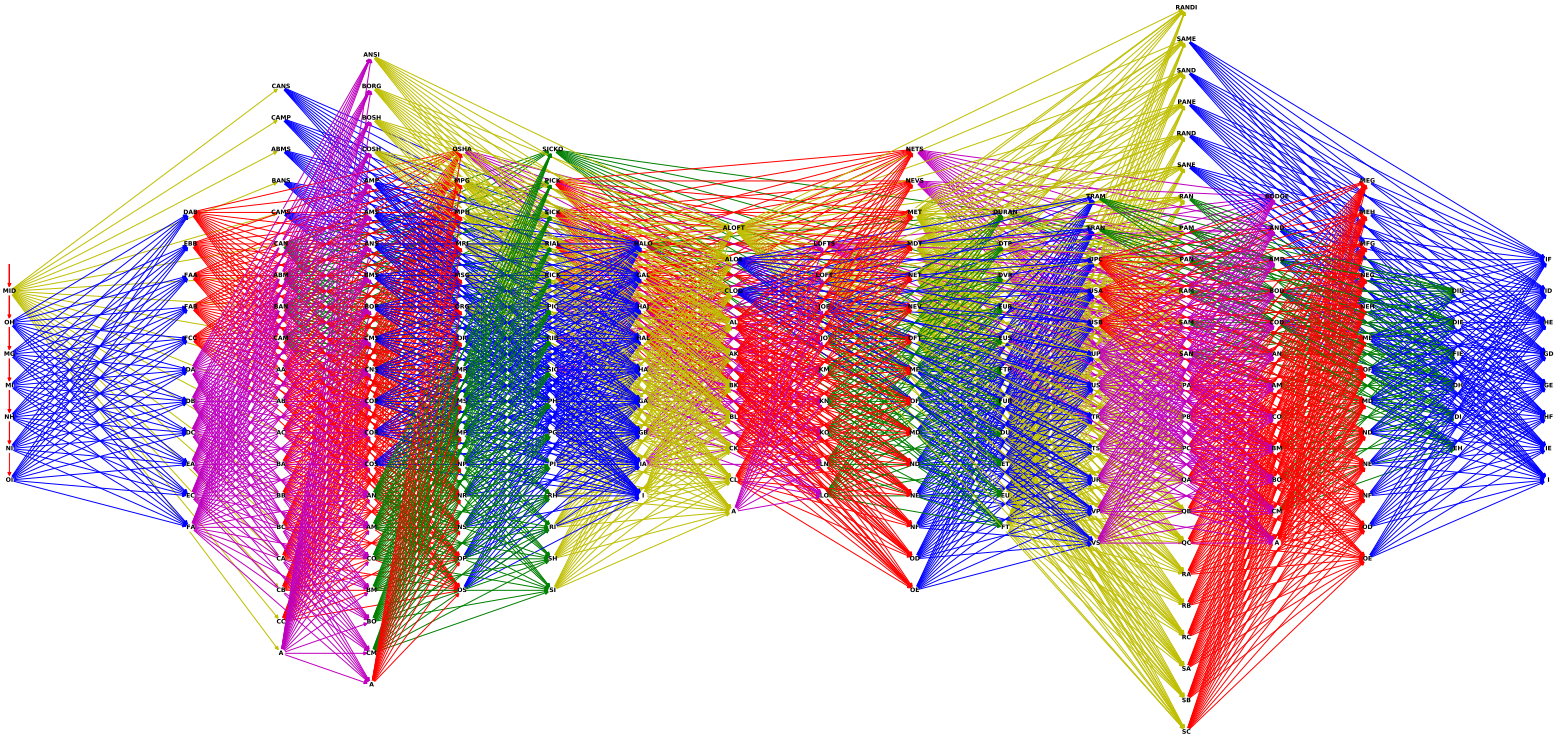
[2] Jonathan Safran Foer. *Extremely Loud & Incredibly Close*. Penguin Books, 2006.

[3] ISO/IEC 9995-8:2009. Information technology Keyboard layouts for text and office systems Part 8: Allocation of letters to the keys of a numeric keypad. Technical report, 2015. `https://www.iso.org/standard/51641.html`.

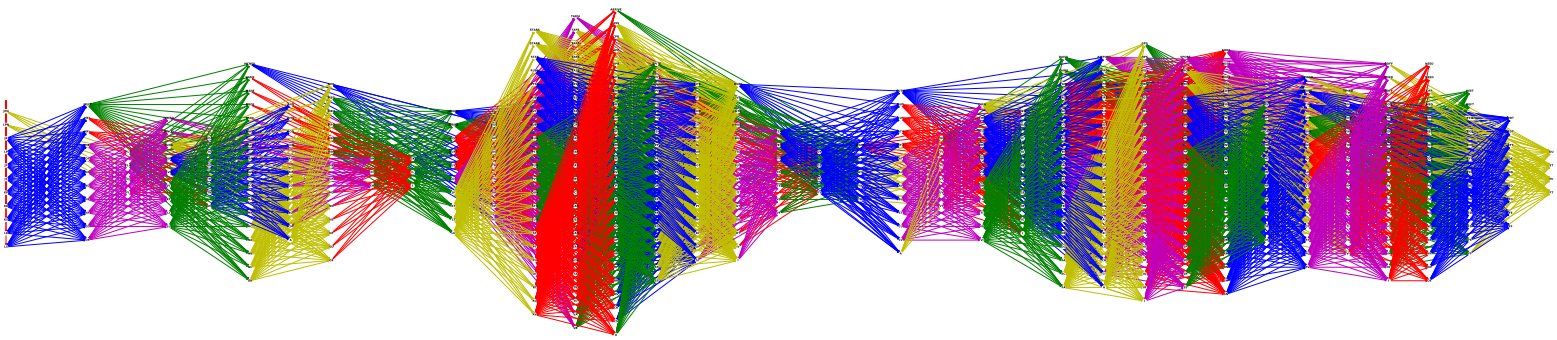
File `sentence1.pdf`



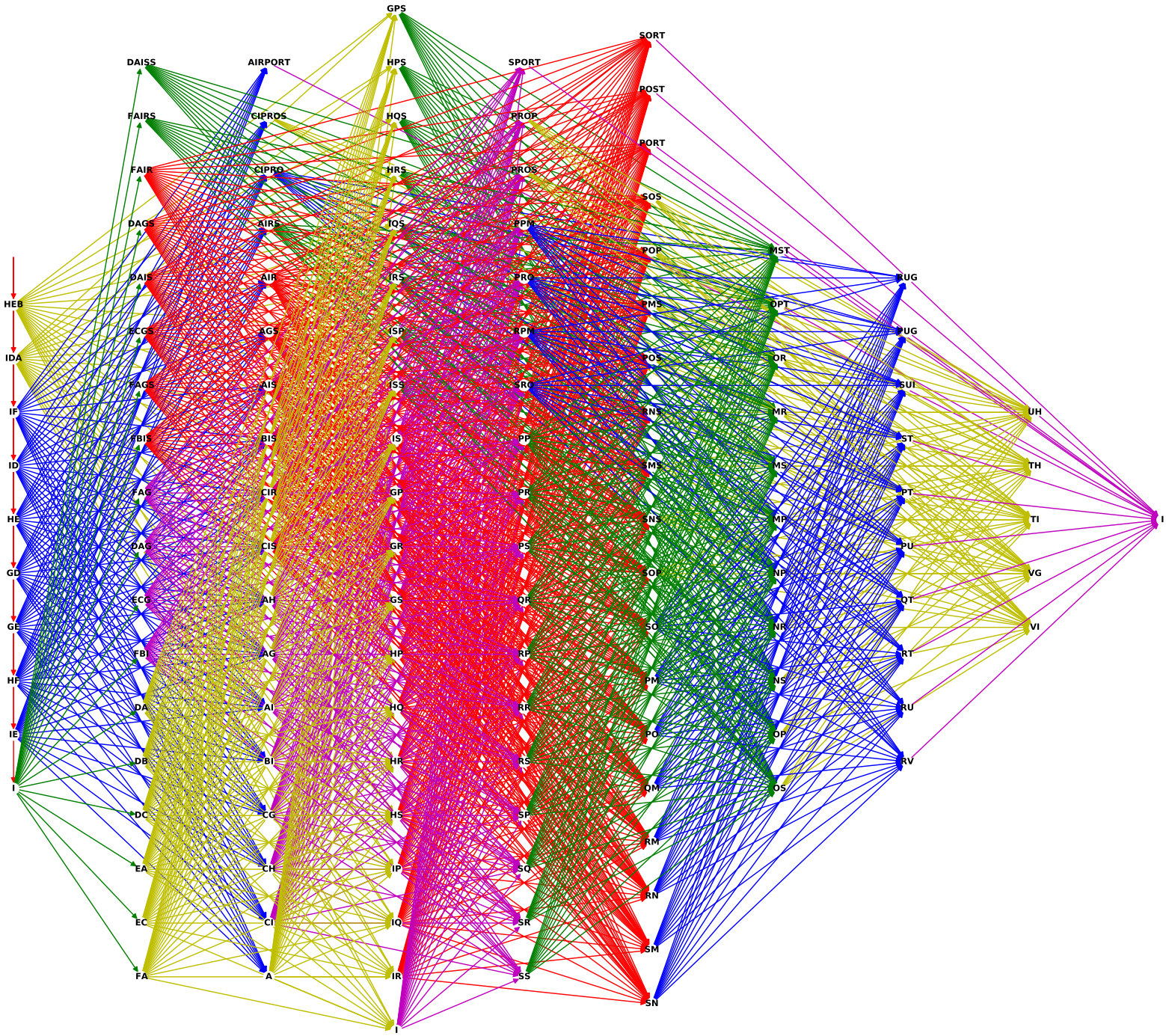
File `sentence2.pdf`



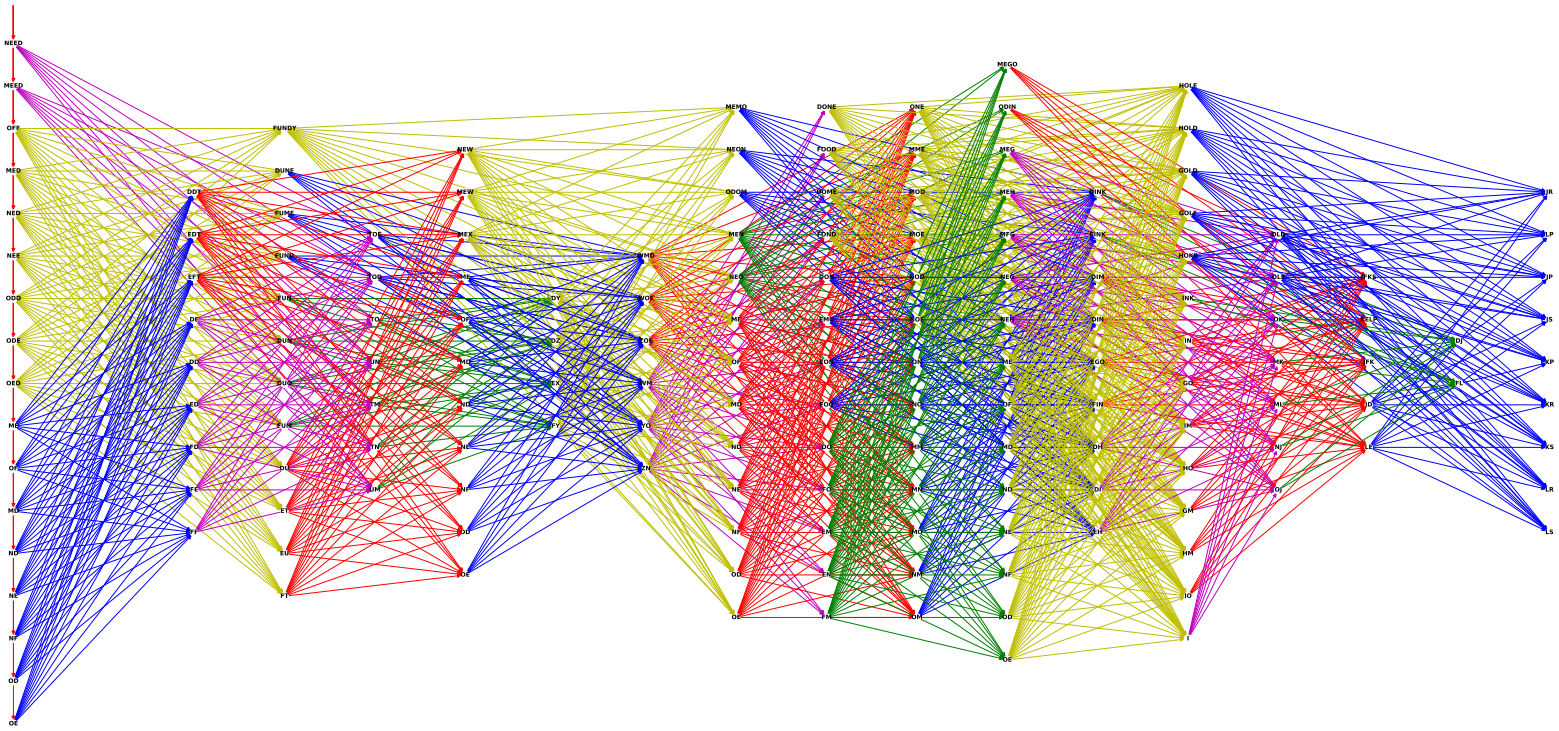
File `sentence3.pdf`



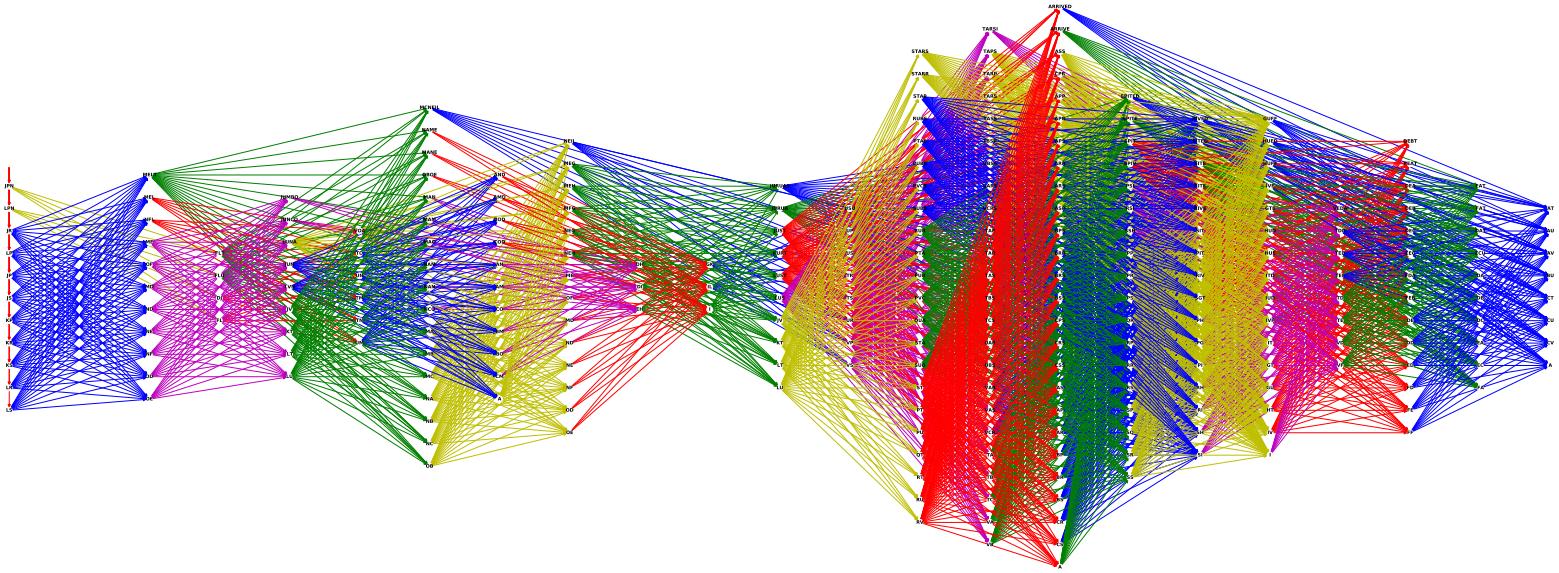
File `sentence4.pdf`



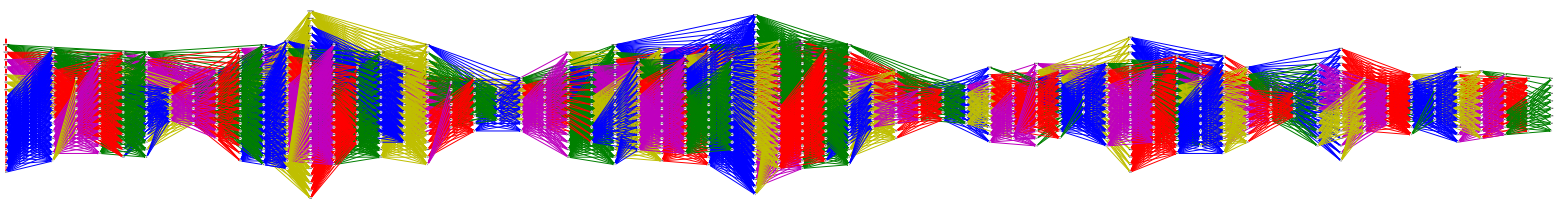
File `sentence5.pdf`



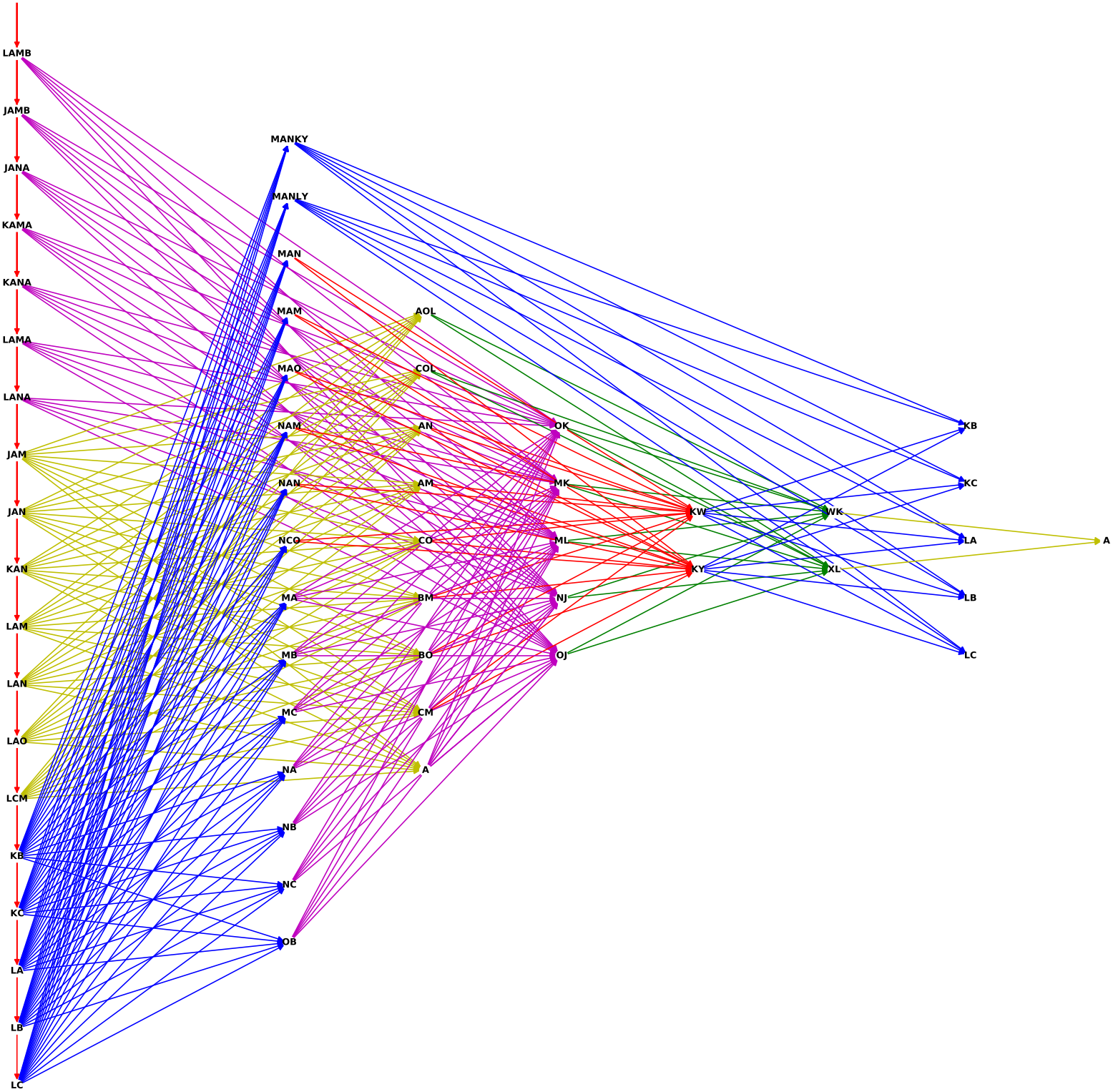
File `sentence6.pdf`



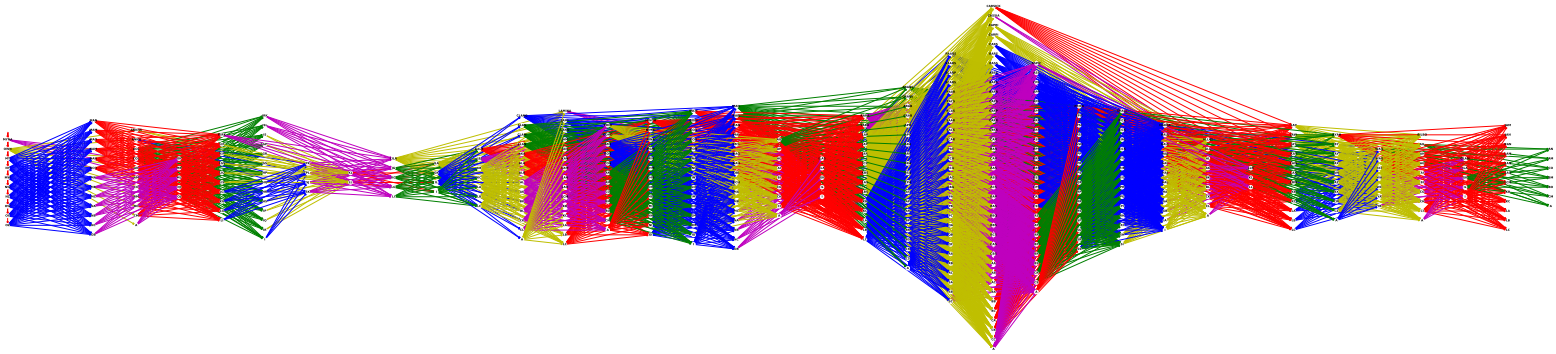
File `sentence7.pdf`



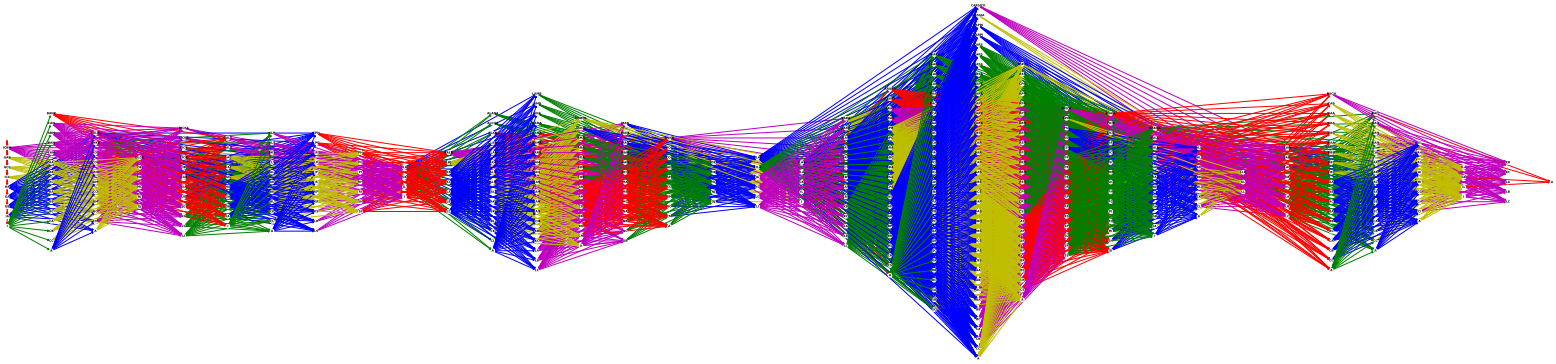
File `sentence8.pdf`



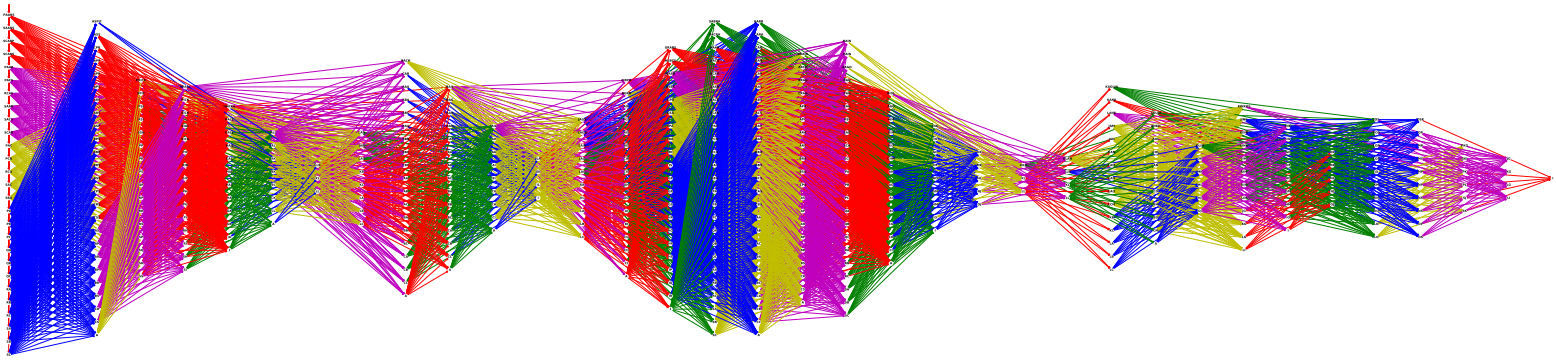
File `sentence9.pdf`



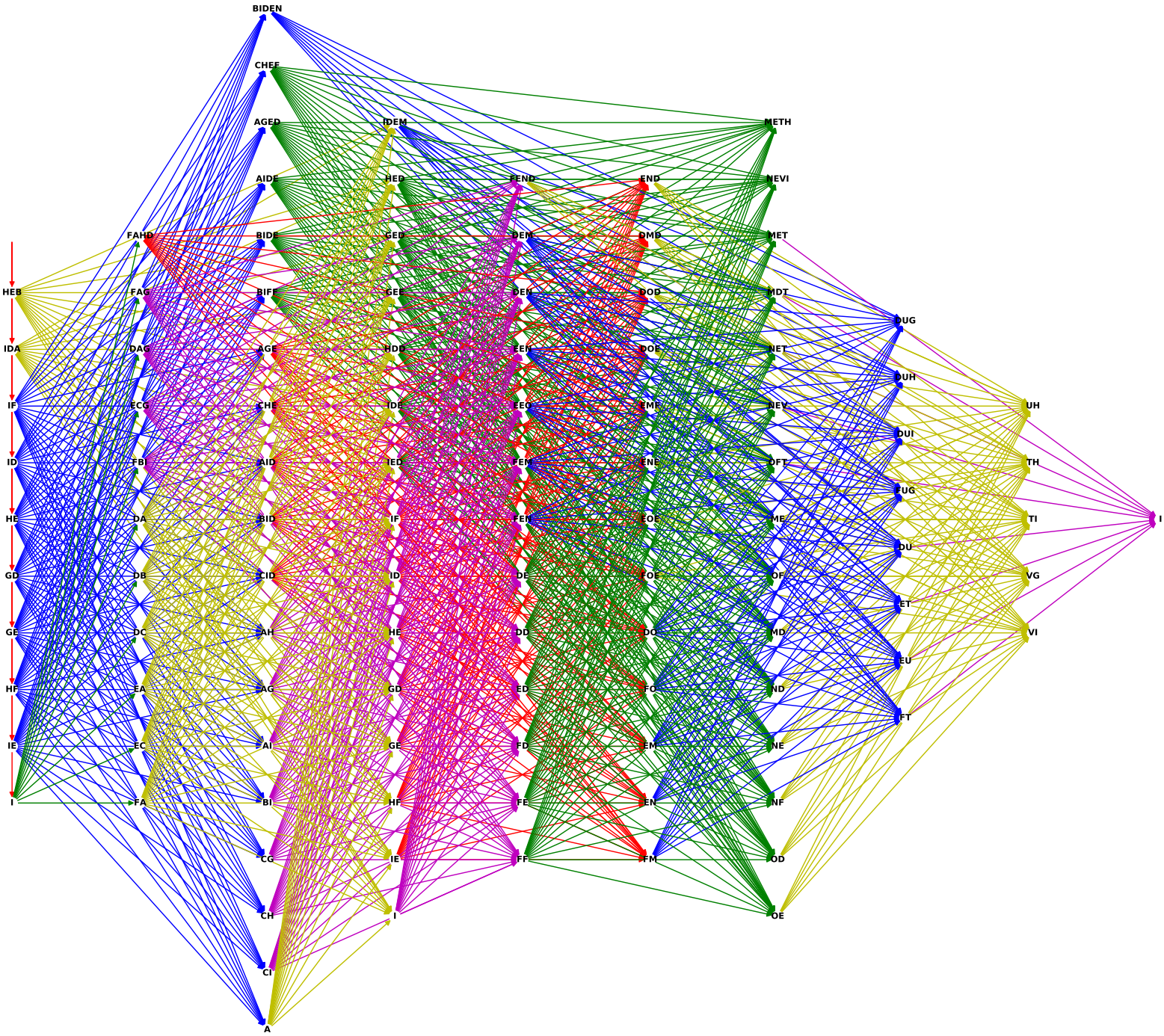
File `sentence10.pdf`



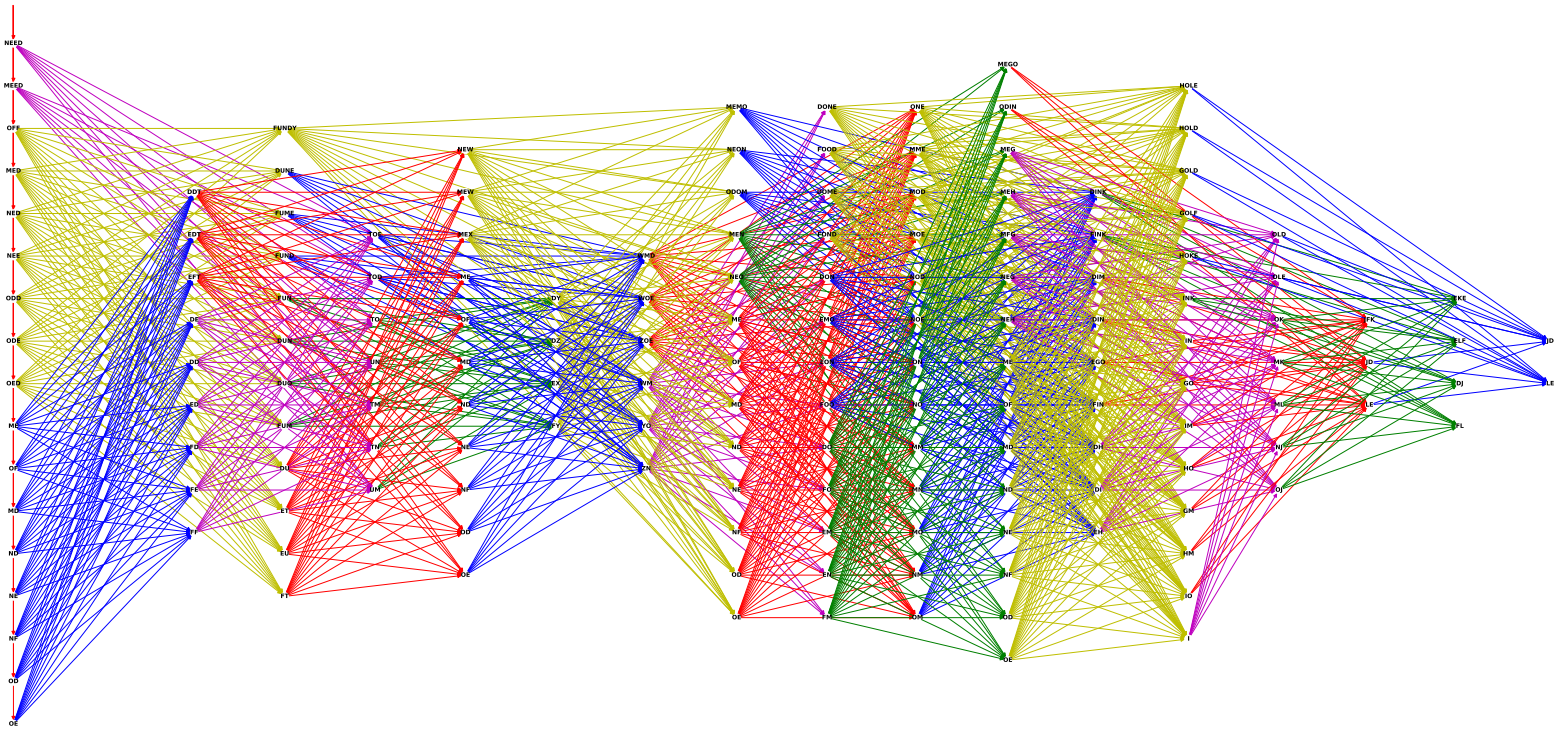
File `sentence11.pdf`



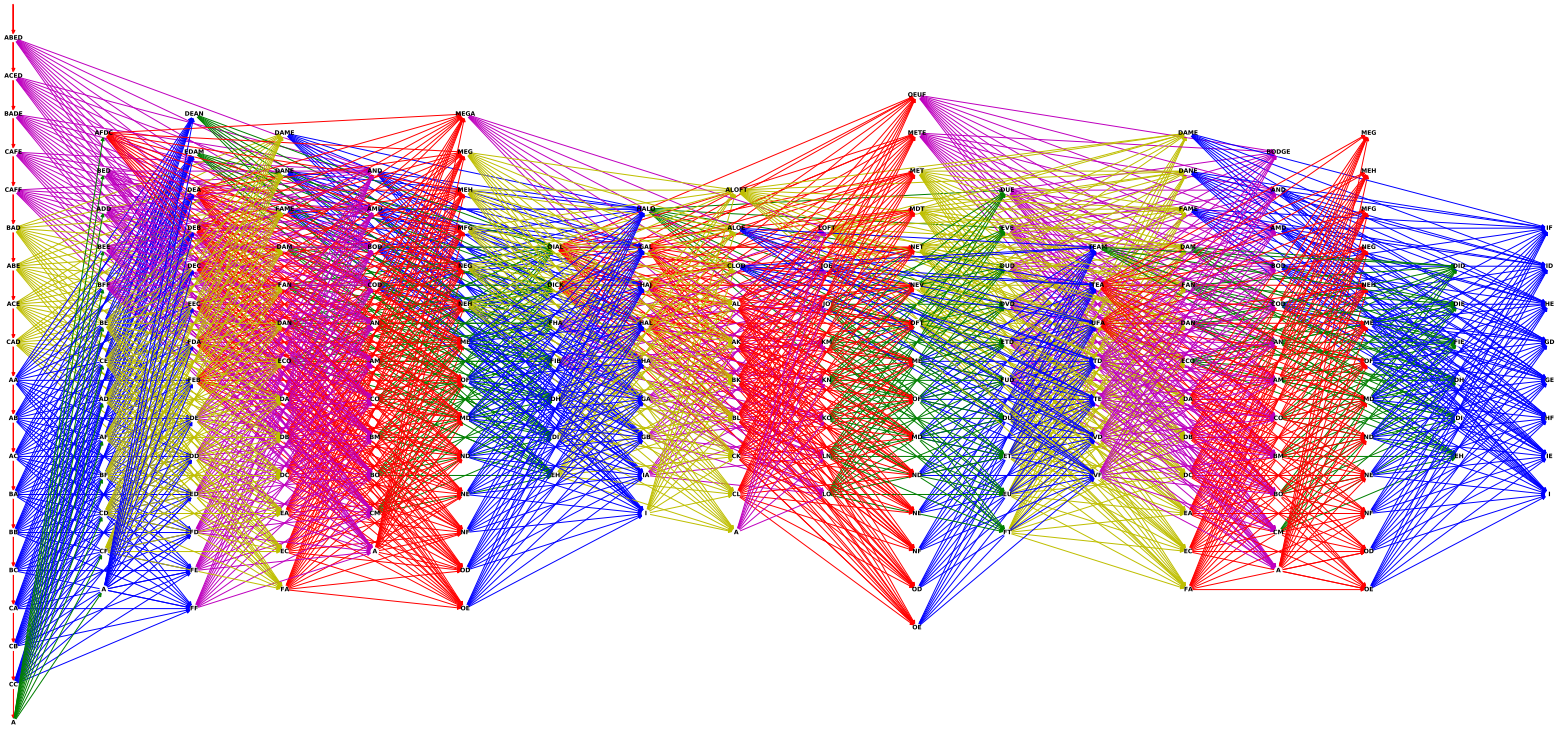
File `sentence12.pdf`



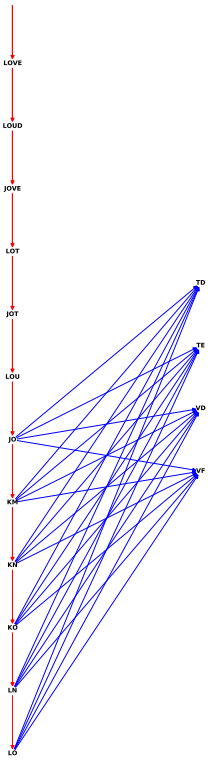
File `sentence13.pdf`



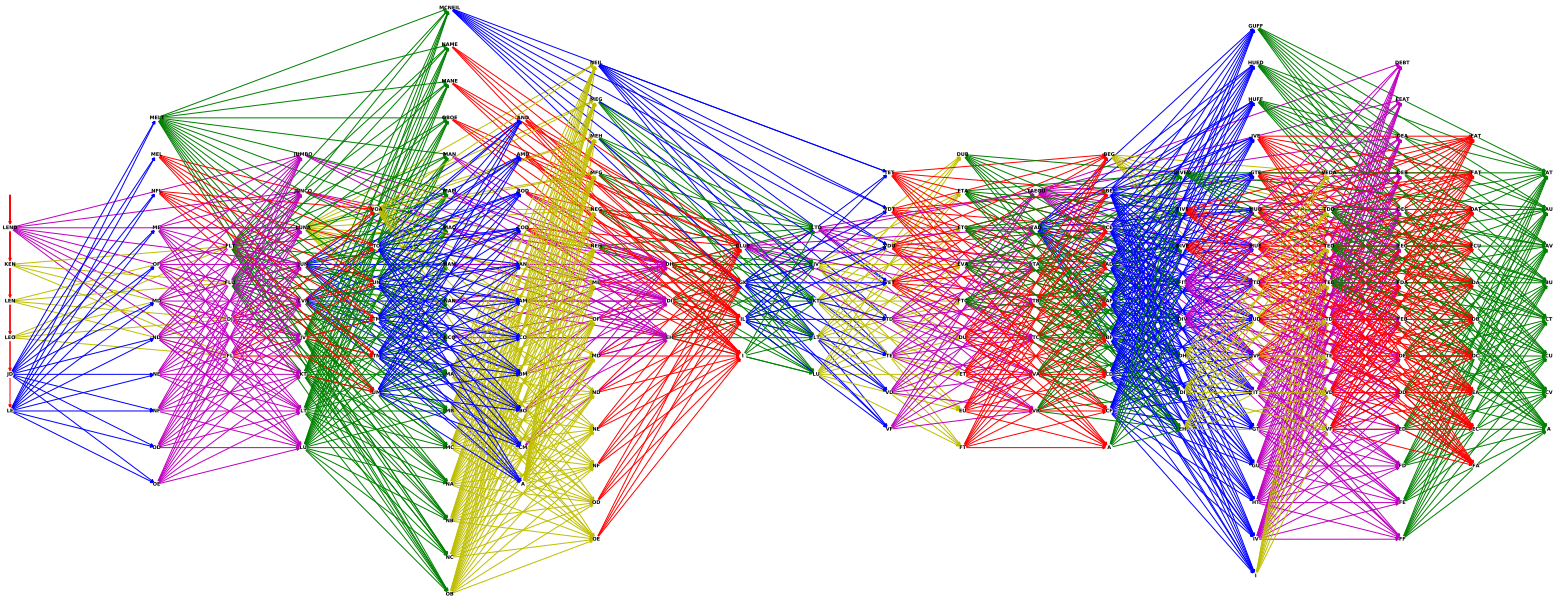
File `sentence14.pdf`



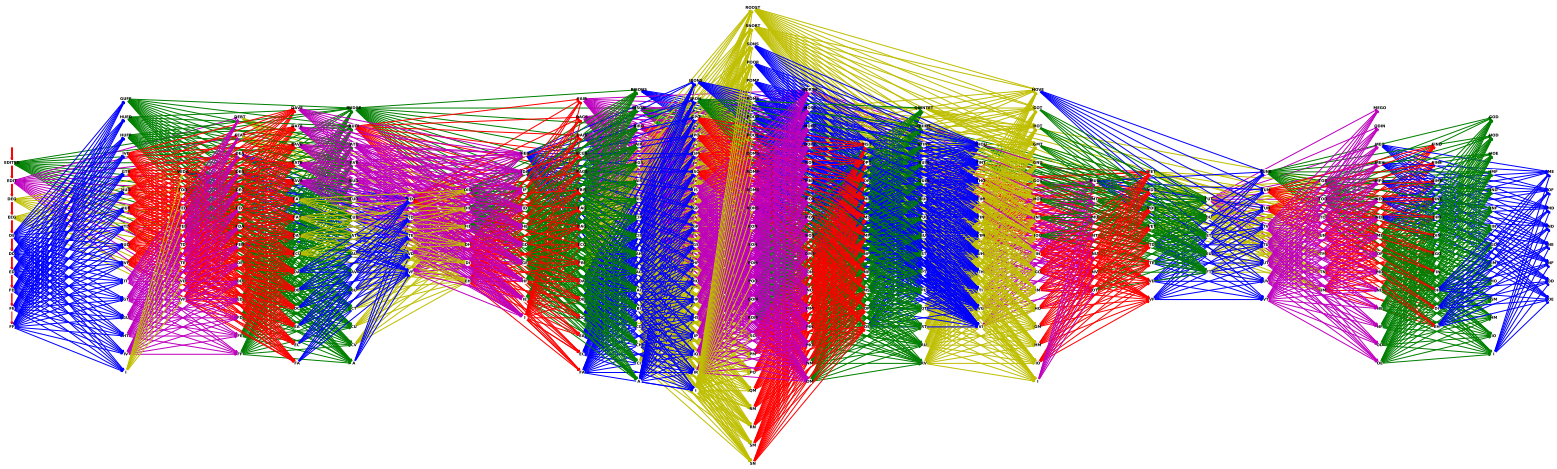
File `sentence15.pdf`



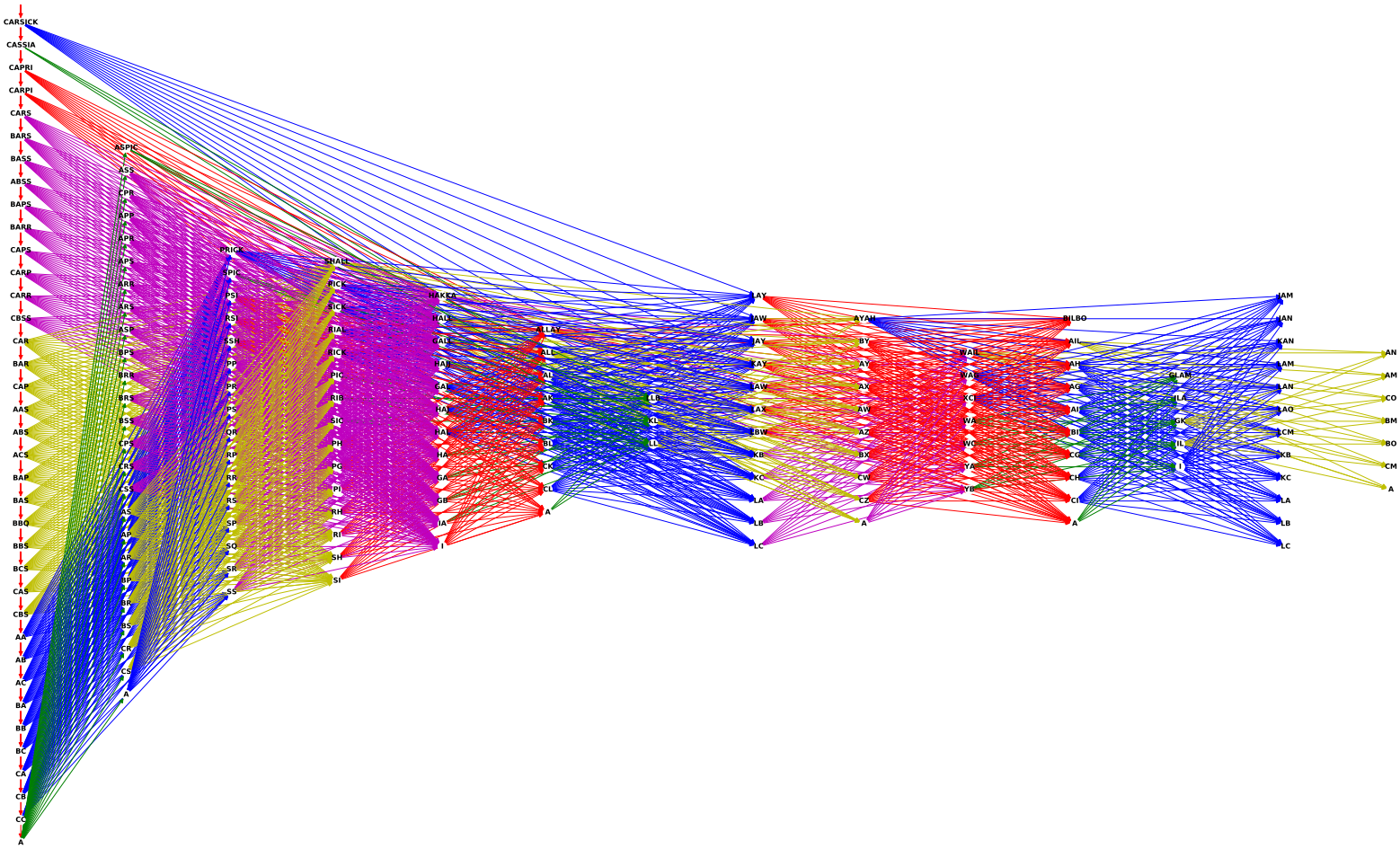
File `sentence16.pdf`



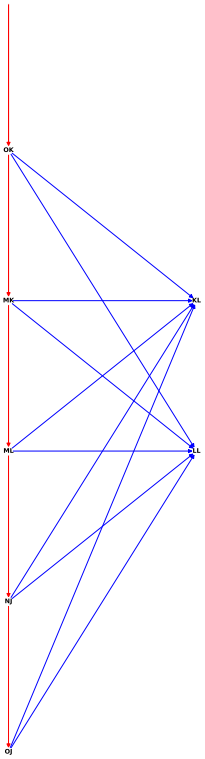
File `sentence17.pdf`



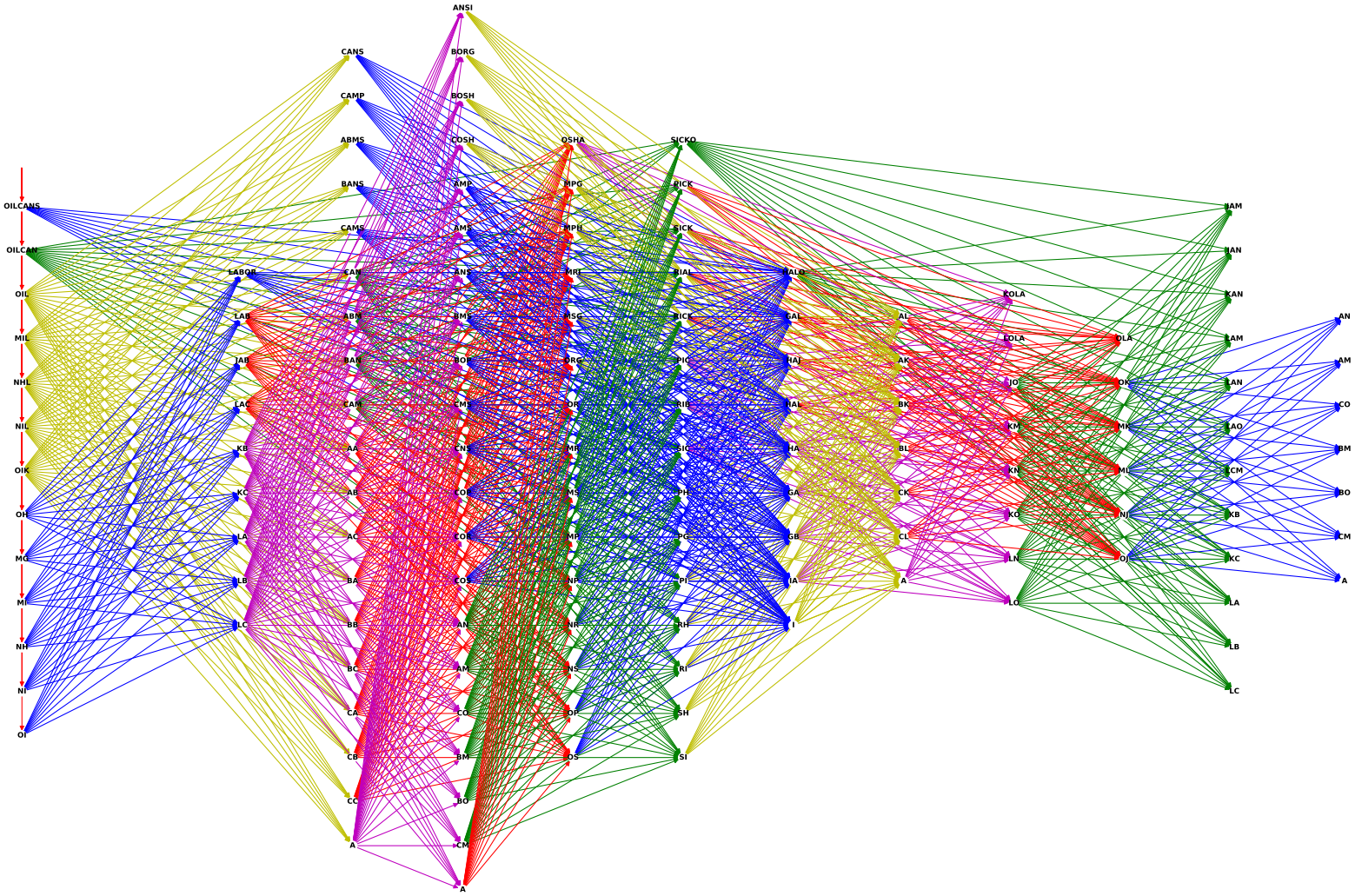
File `sentence18.pdf`



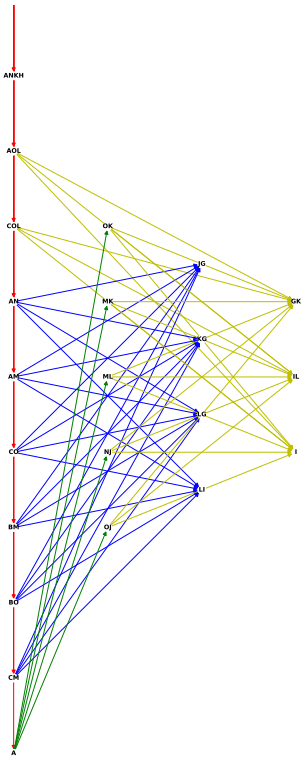
File `sentence19.pdf`



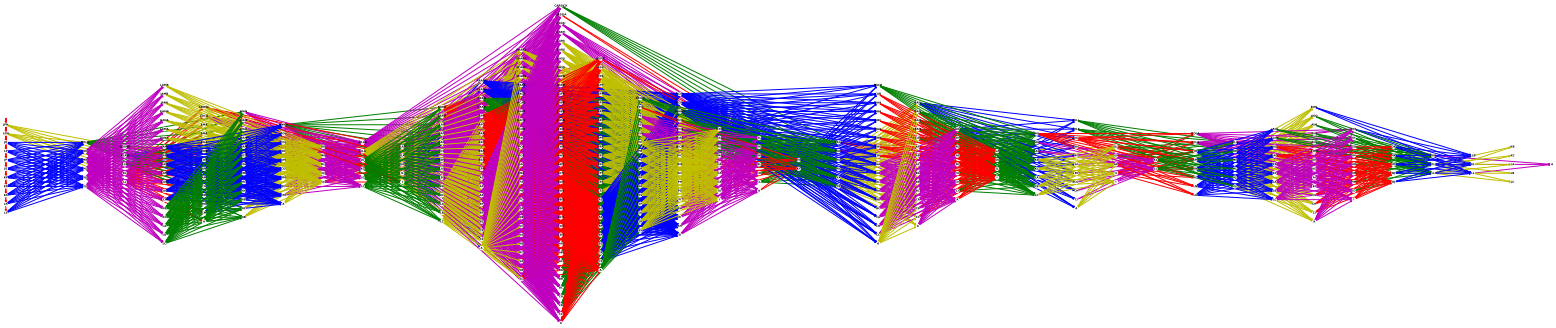
File `sentence20.pdf`



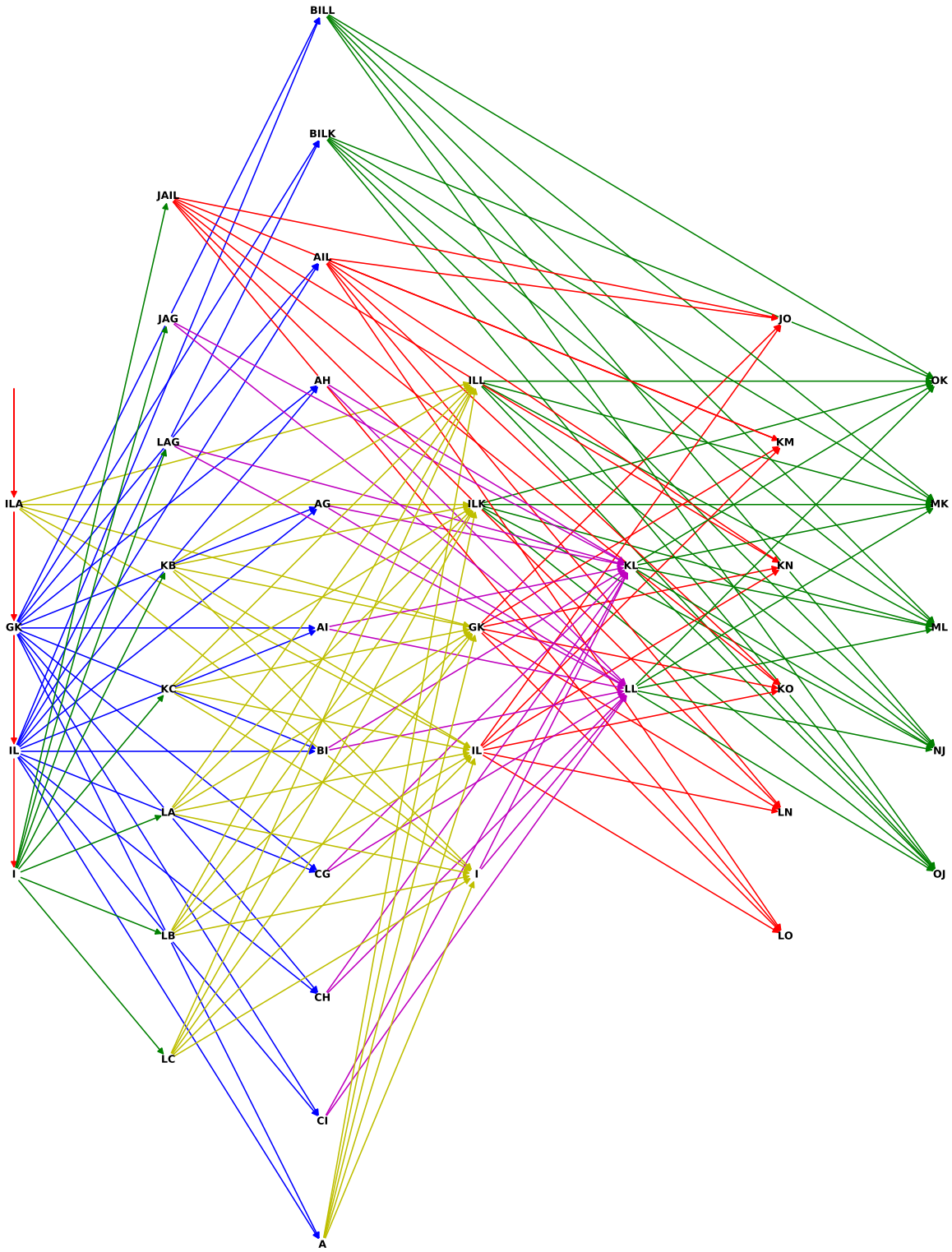
File `sentence21.pdf`



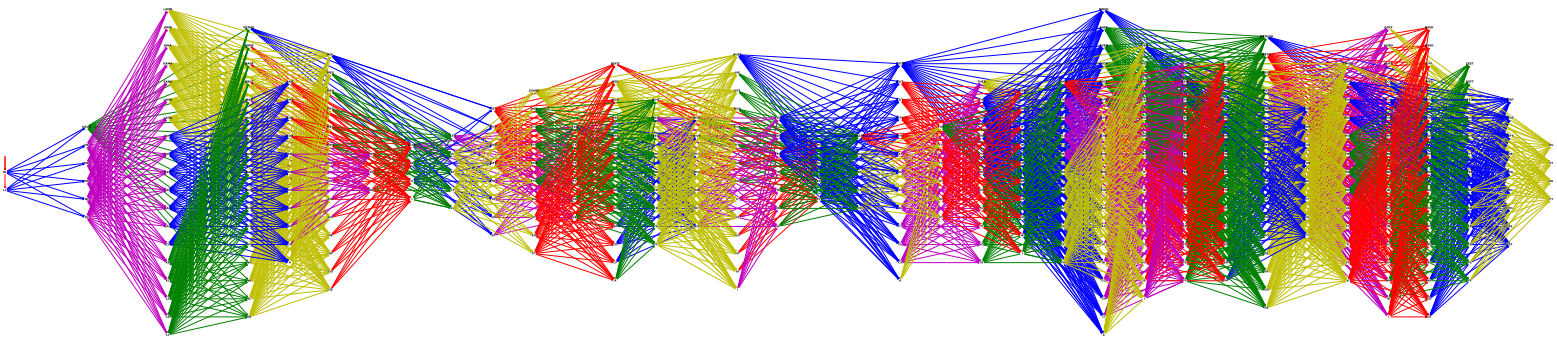
File `sentence22.pdf`



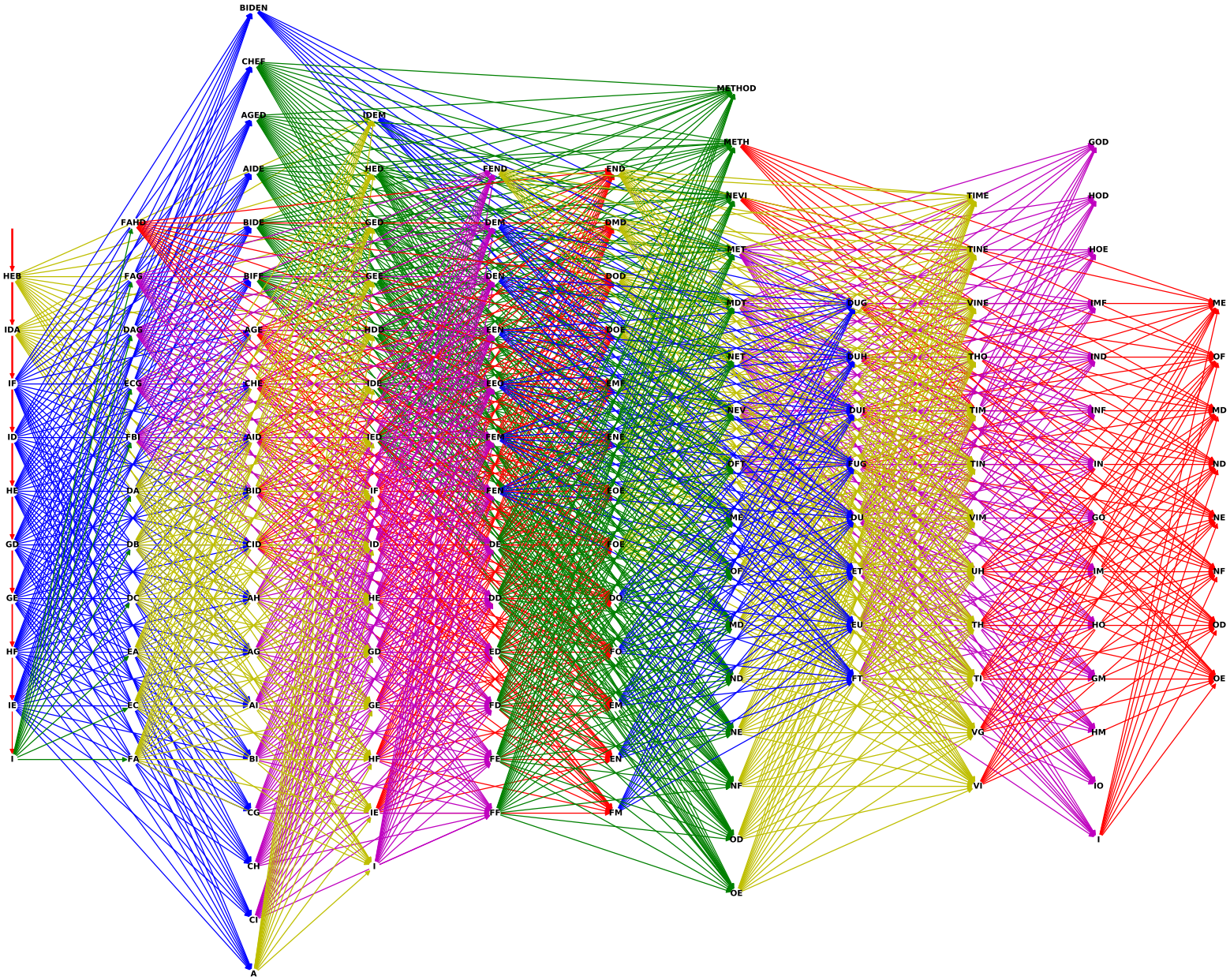
File `sentence23.pdf`



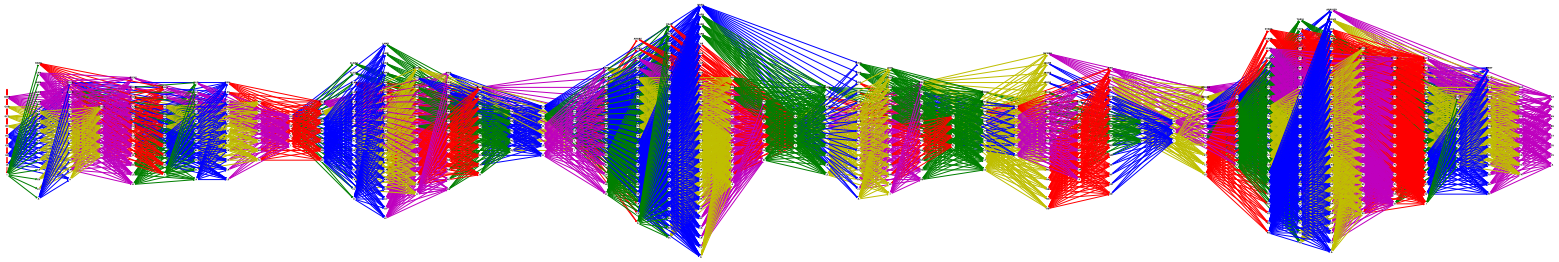
File `sentence24.pdf`



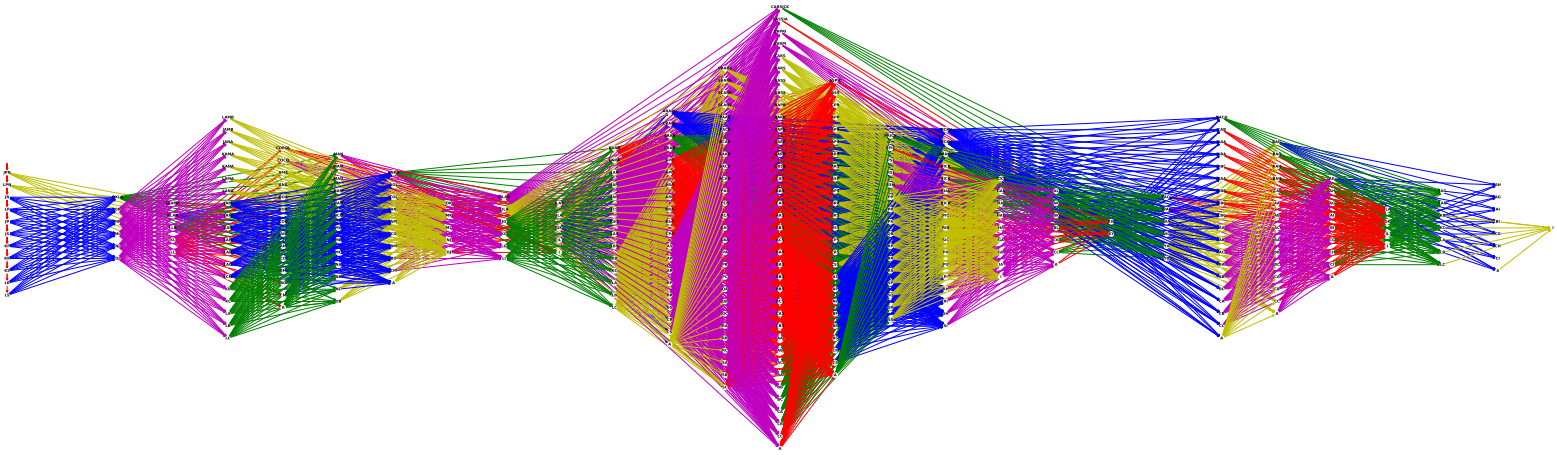
File `sentence25.pdf`



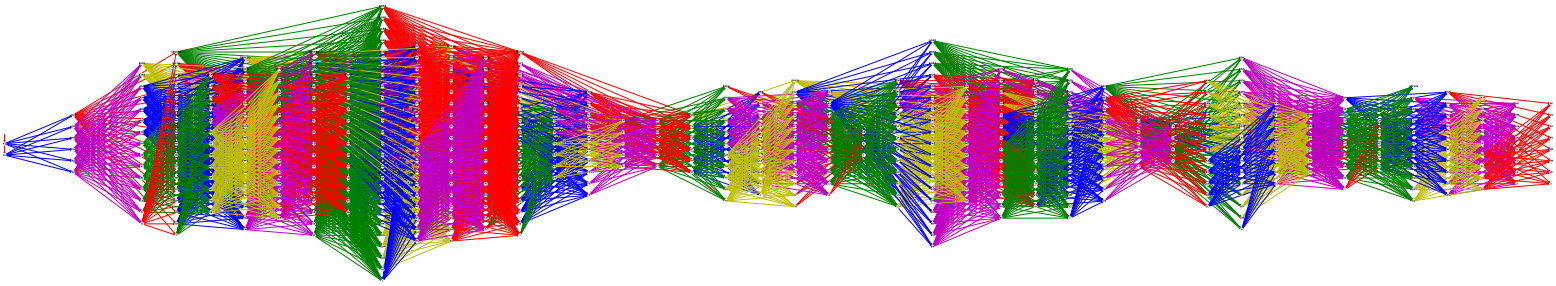
File `sentence26.pdf`



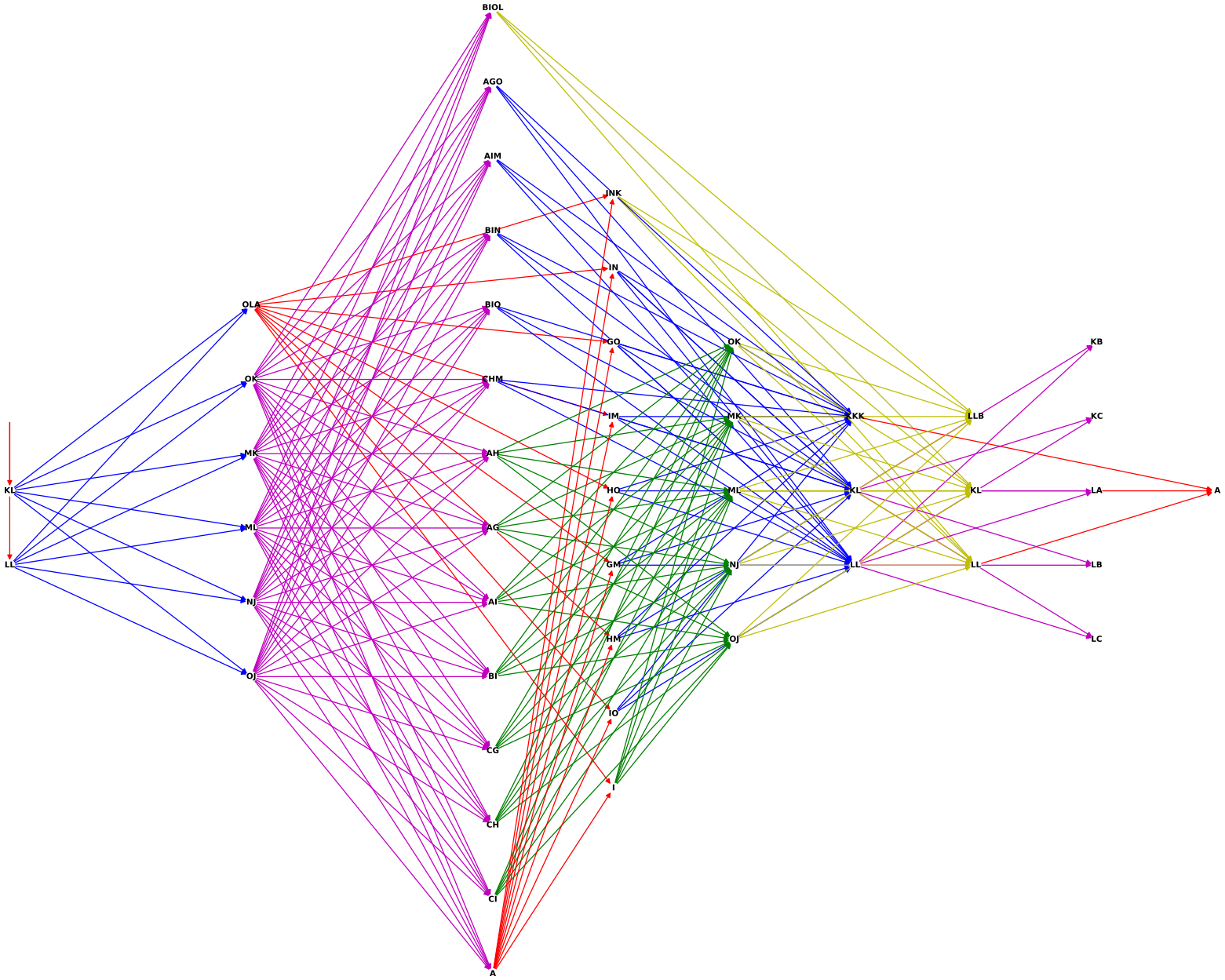
File `sentence27.pdf`



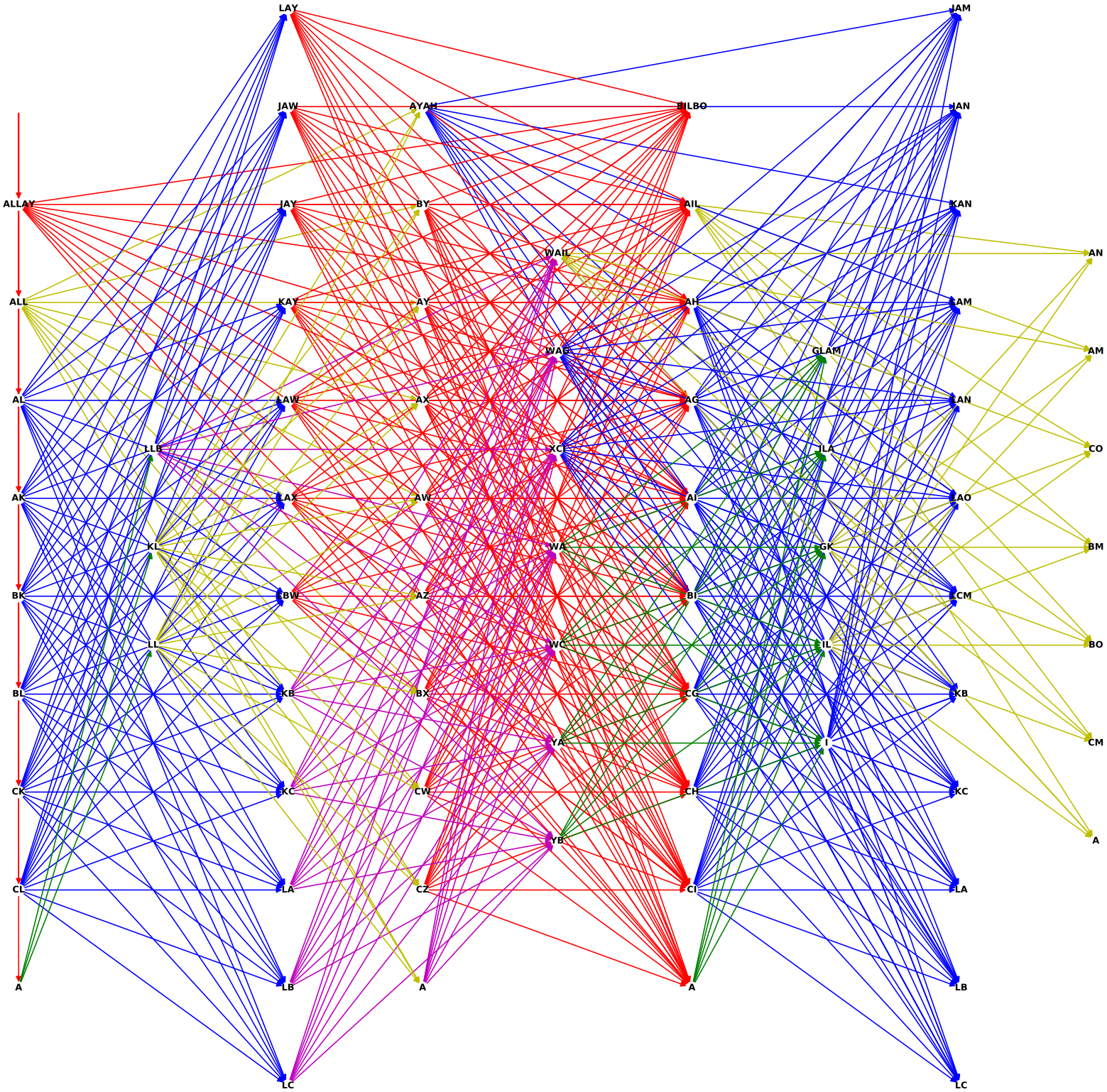
File `sentence28.pdf`



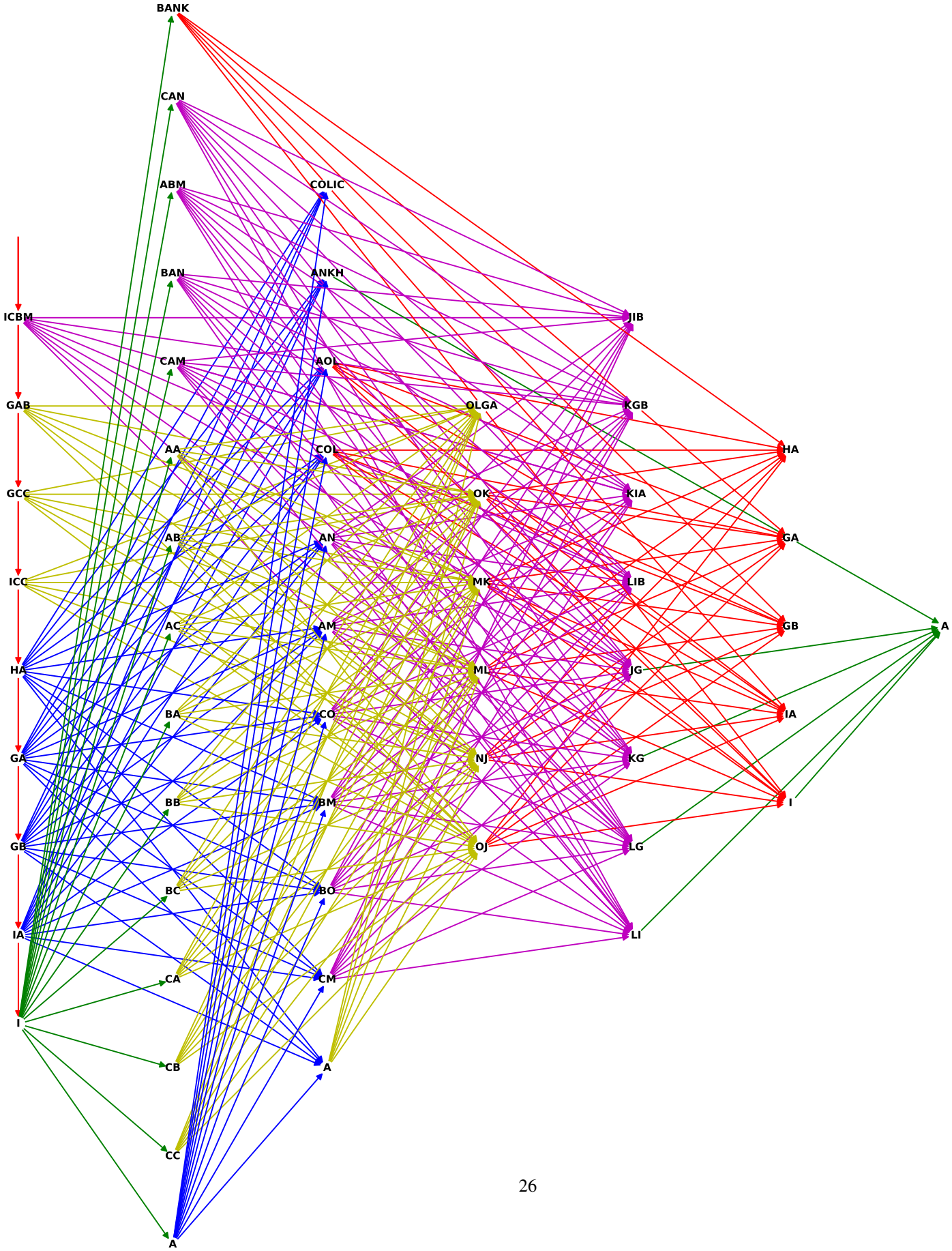
File `sentence29.pdf`



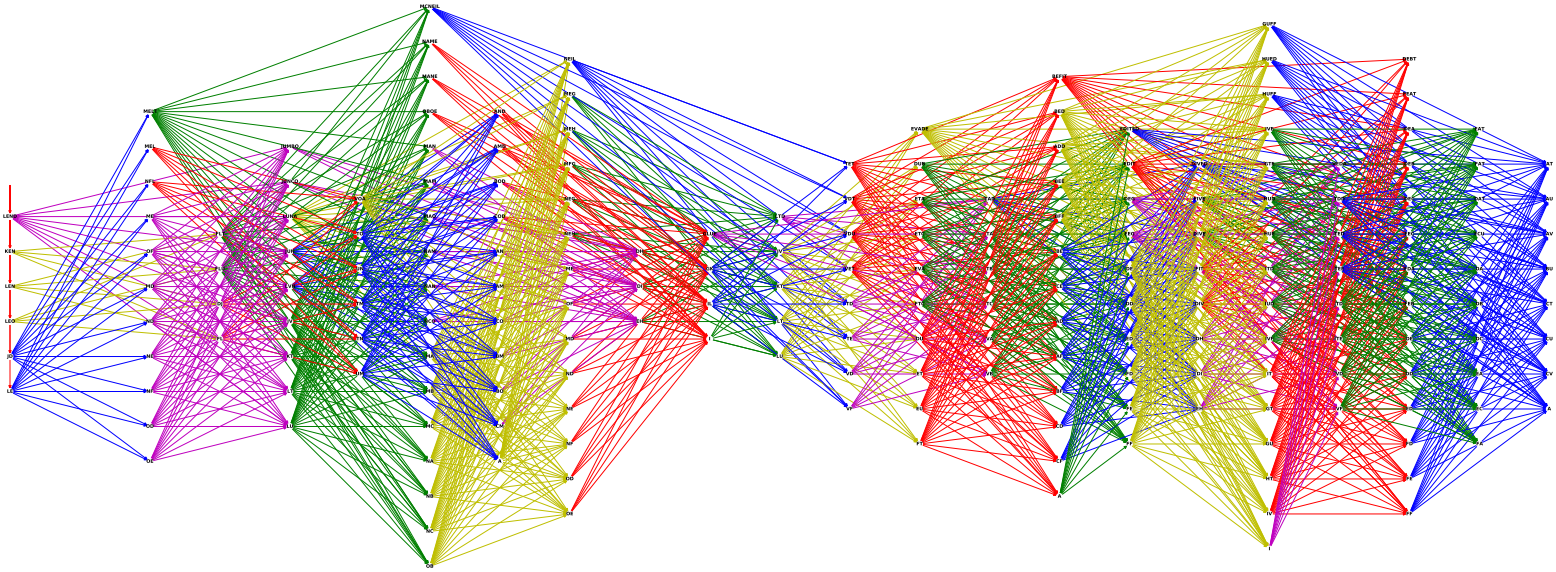
File `sentence30.pdf`



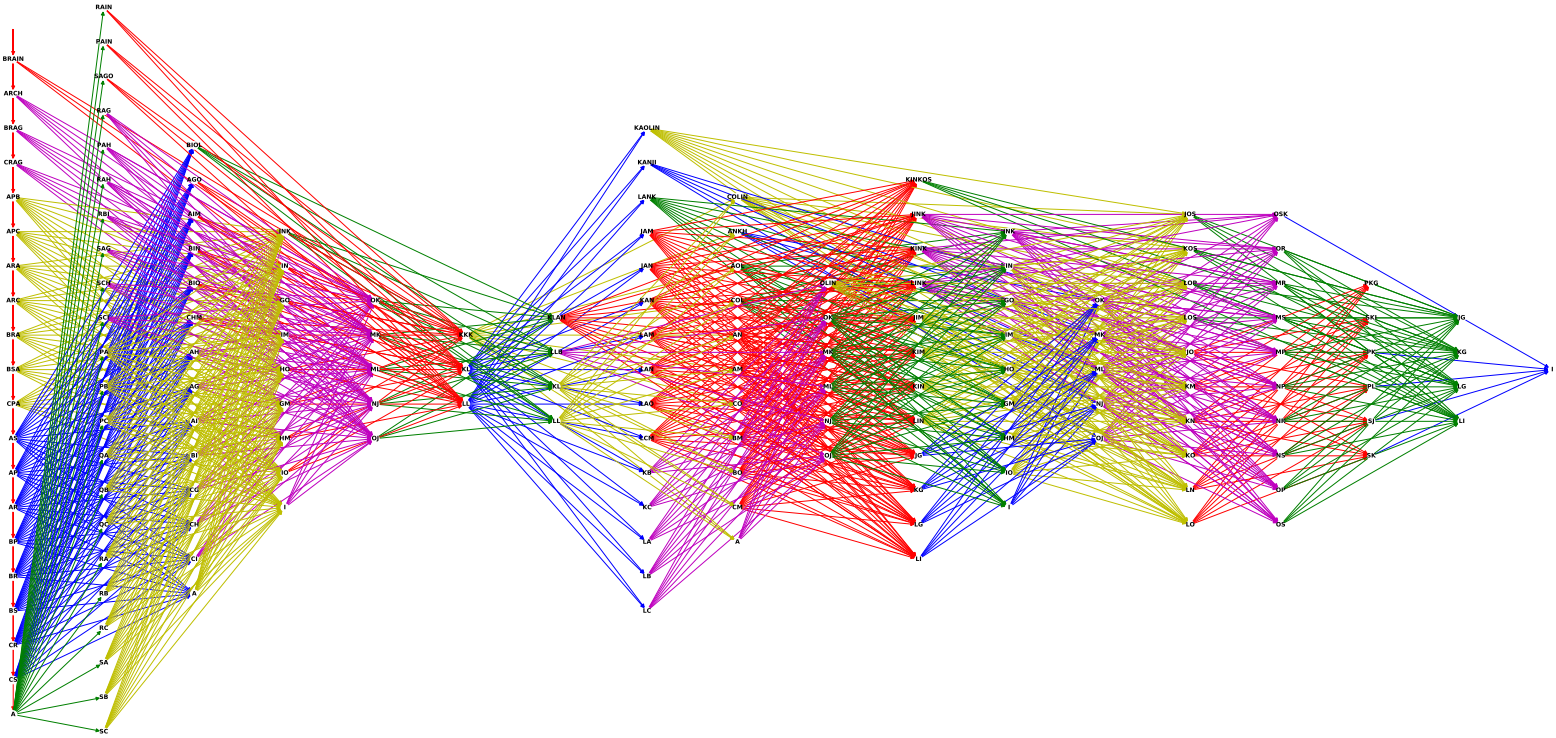
File `sentence31.pdf`



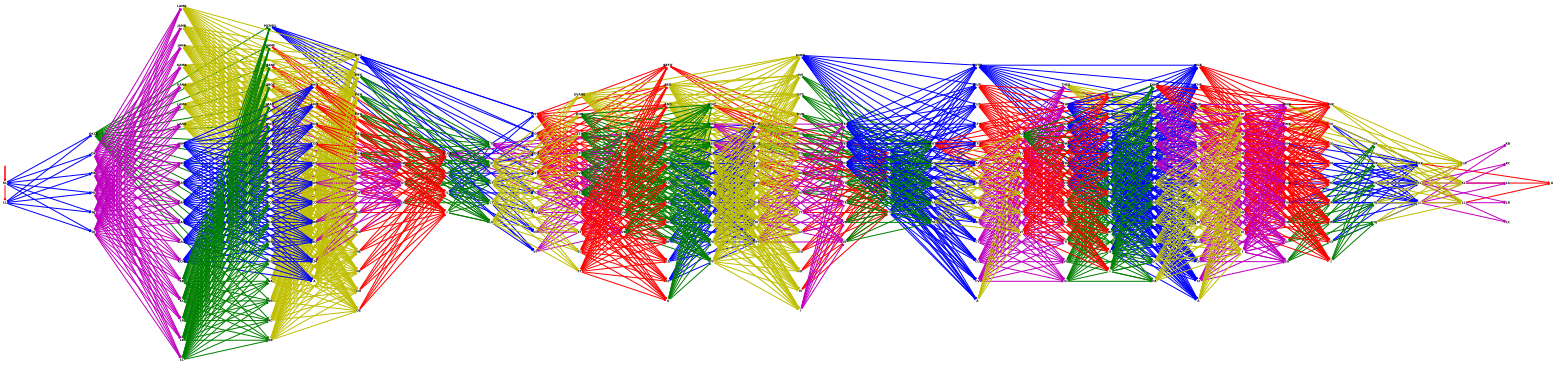
File `sentence32.pdf`



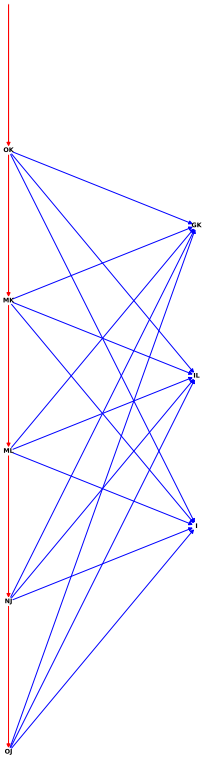
File `sentence33.pdf`



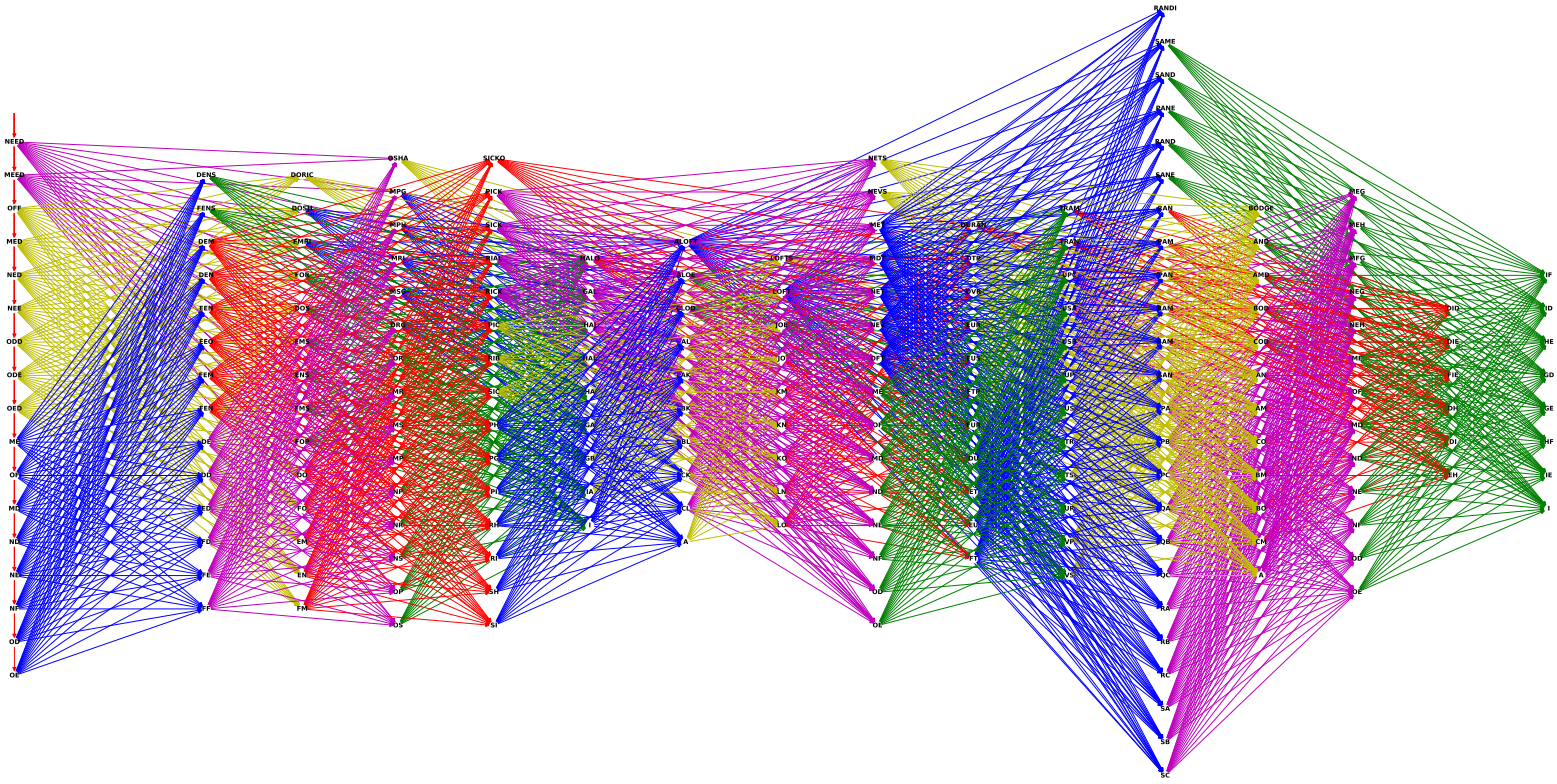
File `sentence34.pdf`



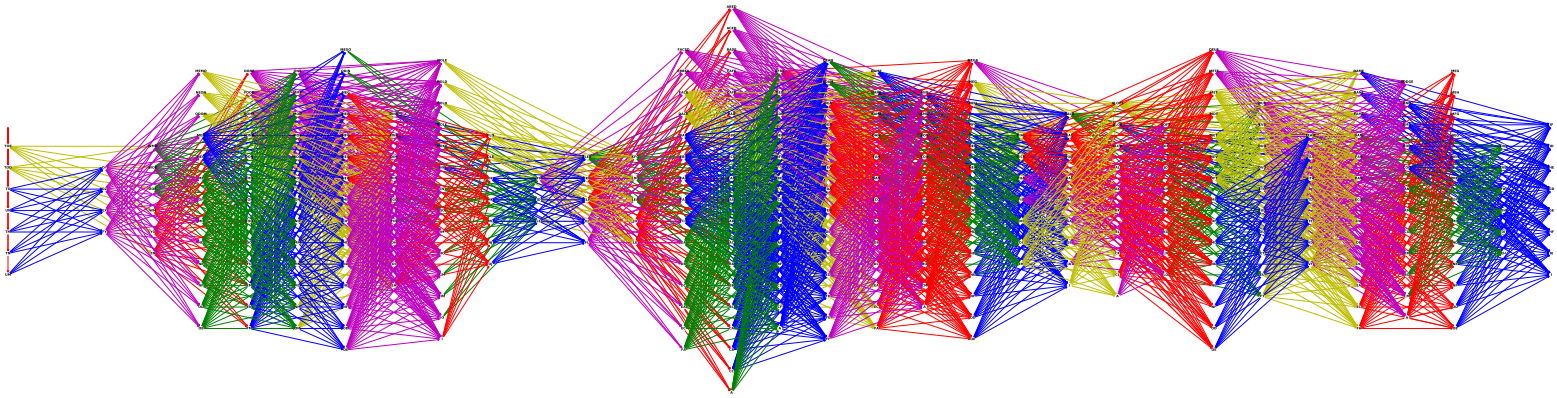
File `sentence35.pdf`



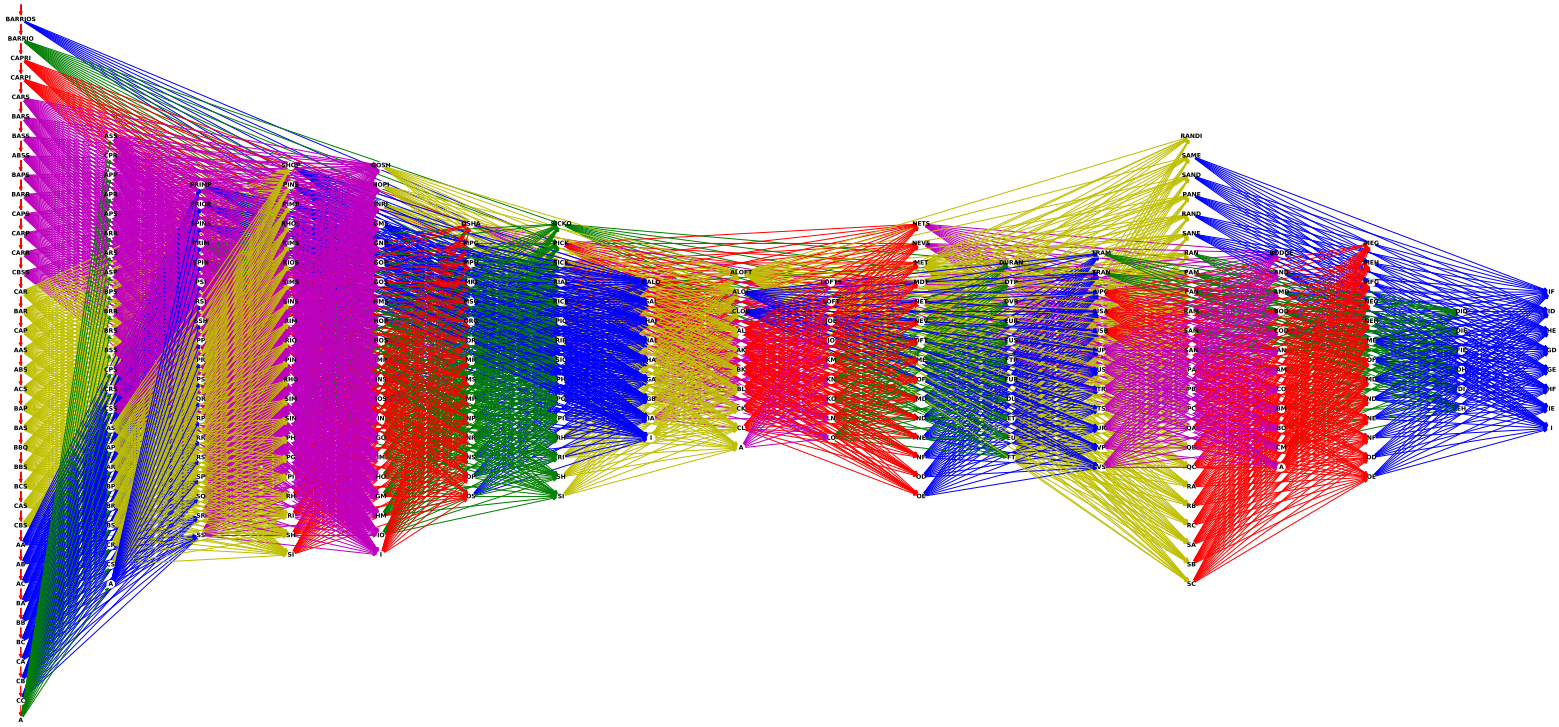
File `sentence37.pdf`



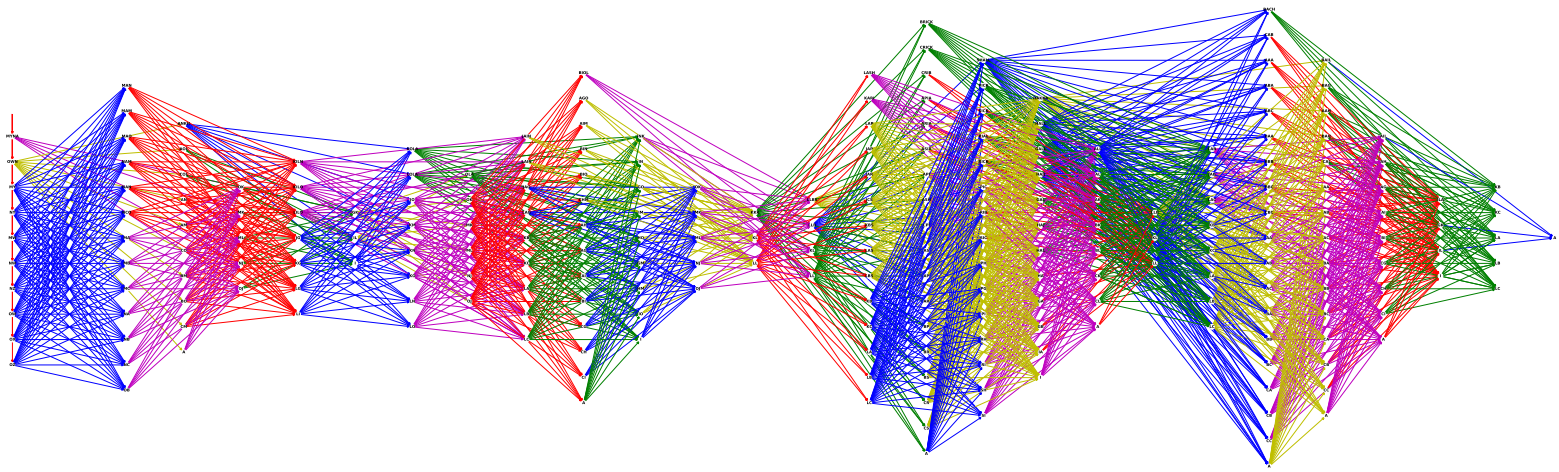
File `sentence38.pdf`



File `sentence39.pdf`



File `sentence40.pdf`



A The Python Code

```
import io, os
import networkx as nx
import matplotlib.pyplot as plt

OCCUR={}
PAIRS={}

mycolor=[]
```

```
color_matches=['r','g','b','y','m']
for i in range(256):
    mycolor.append(color_matches[(i % len(color_matches))])

f=open("tout-u.xml",'r')
for ligne in f:
    words=ligne.split()
    for w in words:
        if w not in OCCUR:
            OCCUR[w]=1
        else:
            OCCUR[w]+=1
    for i in range(len(words)-1):
        if words[i] not in PAIRS:
            PAIRS[words[i]]={}
        if words[i+1] not in PAIRS[words[i]]:
            (PAIRS[words[i]])[words[i+1]]=1
        else:
            (PAIRS[words[i]])[words[i+1]]+=1
f.close()

linecounter=0
counter=0
ENIGMAS=[]
ENIGMACOUNTER={}
f=open("CORPUS.txt",'r')
for ligne in f:
    ligne=ligne[:-2]
    linecounter+=1
    if ligne not in ENIGMACOUNTER.keys():
        counter+=1
        ENIGMAS.append(ligne)
        ENIGMACOUNTER[ligne]=counter
        print("line ",linecounter," is new, it is sentence ",counter)
    else:
        print("line ",linecounter," is old: ",ENIGMACOUNTER[ligne])

wnumber=0
pnumber=0
for w in OCCUR.keys():
    wnumber+=1
    if w in PAIRS:
        for x in (PAIRS[w]).keys():
            pnumber+=1
            (PAIRS[w])[x] /= OCCUR[w]

PAIRS['0']={}
for w in OCCUR.keys():
    (PAIRS['0'])[w]=1

print(str(wnumber)," words, ",str(pnumber)," word pairs\n")

def tosound(word):
    res=""
    for l in word:
        if l in ['A','B','C']:
            res += '2'
        elif l in ['D','E','F']:
            res += '3'
```

```
        elif l in ['G','H','I']:
            res += '4'
        elif l in ['J','K','L']:
            res += '5'
        elif l in ['M','N','O']:
            res += '6'
        elif l in ['P','Q','R','S']:
            res += '7'
        elif l in ['T','U','V']:
            res += '8'
        elif l in ['W','X','Y','Z']:
            res += '9'
    return res

HASH={}
maxlen=0
for w in OCCUR.keys():
    h=tosound(w)
    if len(h) > maxlen:
        maxlen=len(h)
    if h not in HASH:
        HASH[h]=[w,]
    else:
        (HASH[h]).append(w)

print(str(maxlen)," = maxlen, ",str(len(HASH))," hash keys\n")

for ENIGM in ENIGMACOUNTER.keys():
    if len(ENIGM)>2:
        G = nx.DiGraph(directed=True)
        QUEUE=[]
        G.add_node(('',0),ord=0,color=mycolor[0])

        for N in range(maxlen,0,-1):
            if N<=len(ENIGM):
                if ENIGM[:N] in HASH:
                    for w in HASH[ENIGM[:N]]:
                        QUEUE.append((w,0,('0',0)))

        NOT_FINAL=set()
        SUCCESSOR=set()
        while len(QUEUE)>0:
            (a,b,c) = QUEUE.pop(0)
            if (a,b) not in G:
                G.add_node((a,b),ord=b,color=mycolor[b])
                for N in range(maxlen,0,-1):
                    if N<=len(ENIGM[len(a)+b:]):
                        if ENIGM[len(a)+b:len(a)+b+N] in HASH:
                            for w in HASH[ENIGM[len(a)+b:len(a)+b+N]]:
                                QUEUE.append((w,len(a)+b,(a,b)))

        for (a,b) in G:
            for (aa,bb) in G:
                if len(aa)+bb==b:
                    G.add_edge((aa,bb),(a,b),color=mycolor[b])
```

```
maxlevel=0
for g in G:
    if g[1]>maxlevel:
        maxlevel=g[1]



ecolors = nx.get_edge_attributes(G,'color').values()
ncolors = nx.get_node_attributes(G,'color').values()

plt.figure(figsize=(int(2*maxlevel),16))
plt.tight_layout()
pos = nx.multipartite_layout(G, subset_key='ord',scale=5)
nx.draw(G, pos=pos, with_labels=True,labels={x:x[0] for x in G},edge_color=ecolors,node_color="
plt.savefig("sentence"+str(ENIGMACOUNTER[ENIGM])+".pdf")
plt.close()
print("Just wrote file "+"sentence"+str(ENIGMACOUNTER[ENIGM])+".pdf")
os.system("pdfcrop "+"sentence"+str(ENIGMACOUNTER[ENIGM])+".pdf")
```