In [56]:
```
import nltk
import os
import re
import string
from IPython.display import Image
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
import gensim
import numpy as np
```

In [6]:
```
input_directory = "1999 Hepth CTM LDA Topics fulltexts/1999 Hepth CTM LDA Topics fullte
```

In [7]:
```
input_docs_id = os.listdir("1999 Hepth CTM LDA Topics fulltexts/1999 Hepth CTM LDA Topi
```

In [8]:
```
for doc_id in input_docs_id:
    if not doc_id.endswith(".txt"):
        input_docs_id.remove(doc_id)
```

In [9]:
```
input_docs_content = []
```

In [10]:
```
for doc_id in input_docs_id[:100]:
    with open(input_directory + "/" + doc_id,"r") as f:
        input_docs_content.append(f.read())
```

In [11]:
```
len(input_docs_content)
```

Out[11]:
```
100
```

Pre Processing

Text lowercase conversion

In [12]:
```
for i in range(len(input_docs_content)):
    input_docs_content[i] = input_docs_content[i].lower()
```

Remove numbers

In [13]:
```
for i in range(len(input_docs_content)):
    input_docs_content[i] = re.sub(r'\d+', '', input_docs_content[i])
```

Remove Punctuation

In [14]:
```
```

```
translator = str.maketrans('', '', string.punctuation)
for i in range(len(input_docs_content)):
    input_docs_content[i] = input_docs_content[i].translate(translator)
```

Remove WhiteSpace

In [15]:
```
for i in range(len(input_docs_content)):
    input_docs_content[i] = " ".join(input_docs_content[i].split())
```

In [ ]:
```

```

Tokenization

In [16]:
```
feature_terms = []
for i in range(len(input_docs_content)):
    feature_terms.append(nltk.tokenize.word_tokenize(input_docs_content[i]))
```

In [ ]:
```

```

Stop Word removal

In [17]:
```
stop_words = set(nltk.corpus.stopwords.words("english"))

for i in range(len(feature_terms)):
    feature_terms[i] = [word for word in feature_terms[i] if word not in stop_words]
```

remove single and double letter words

In [18]:
```
filtered_feature_term=[]

for feature_term in feature_terms:
    temp_list = []
    for feature in feature_term:
        if not(len(feature)==1 or len(feature)==2):
            temp_list.append(feature)
    filtered_feature_term.append(temp_list)
```

In [19]:
```
feature_terms = filtered_feature_term
```

In [20]:
```
pd.DataFrame(feature_terms)
```

Out[20]:

	0	1	2	3	4	5	6	7
0	yitp	hepth	january	string	junctions	duals	heterotic	string
1	yitp	hepth	january	string	junctions	duals	heterotic	string

	0	1	2	3	4	5	6	7
2	yitp	hepth	january	string	junctions	duals	heterotic	string
3	yitp	hepth	january	string	junctions	duals	heterotic	string
4	ctp	tamu	uprt	january	hepth	exact	absorption	probability
...	...	...	...	...	...	...	...	...
95	nambugoldstone	mechanism	realtime	thermal	eld	theory	department	physics
96	nambugoldstone	mechanism	realtime	thermal	eld	theory	department	physics
97	landau	institute	preprint	topological	interpretation	quantum	numbers	sabulgadaev
98	landau	institute	preprint	topological	interpretation	quantum	numbers	sabulgadaev
99	yangmills	field	solitons	back	faddeev	steklov	mathematical	institute

100 rows × 7492 columns

Stemming

In [21]:
```
# filtered_feature_term=[]

# stemmer = nltk.stem.porter.PorterStemmer()

# for feature_term in feature_terms:
#     filtered_feature_term.append([stemmer.stem(word) for word in feature_term])
```

A. TERM CORRELATION BASED ON CO-OCCURRENCE

In [22]:
```
unique_words = set()
```

In [23]:
```
for feature_term in feature_terms:
    for feature in feature_term:
        unique_words.add(feature)
```

In [24]:
```
len(unique_words)
```

Out[24]:
```
10968
```

In [25]:
```
unique_words = list(unique_words)
```

In [26]:
```
Image(filename='images/R_formula.png')
```

Out[26]:

$$r(T_i, T_j) = \frac{P(T_i \,|T_j) \cdot P(T_j \,|T_i)}{P(T_i \,|T_j) + P(T_j \,|T_i) - P(T_i \,|T_j) \cdot P(T_j \,|T_i)}$$

In [27]:
```
Image(filename='images/R_formula_1.png')
```

Out[27]:

$$P(T_i \,|T_j) = \frac{b}{b+c}$$

$$P(T_j \,|T_i) = \frac{b}{a+b}$$

In [28]:
```
Image(filename='images/R_formula_2.png')
```

Out[28]:

Let a represent the number of items with term T_i but not term T_j in the document corpus, c the number of items with term T_j but not term T_i, and b the number of items with both terms. The term association presented here measures the *interactions between feature terms across the entire multi-document corpus*. A higher value of $r(T_\mathrm{i}, T_\mathrm{j})$ indicates a stronger relationship between T_i and T_j, whereas a lower one represents a weaker relationship.

In [29]:
```
R_matrix=[]
for T_i in unique_words[:50]:
    temp_list = []
    for T_j in unique_words[:50]:
        if T_i == T_j:
            temp_list.append(0)
            continue
        a=0
        b=0
        c=0
        for feature_term in feature_terms:
            if (T_i in feature_term) and not (T_j in feature_term):
                a=a+1
            if (T_j in feature_term) and not (T_i in feature_term):
                c=c+1
            if (T_j in feature_term) and (T_i in feature_term):
                b=b+1
        try:
            ptitj = (b)/(b+c)
        except ZeroDivisionError:
            ptitj = 0
```

```
        try:
            ptjti = (b)/(a+b)
        except ZeroDivisionError:
            ptjti = 0
        try:
            rtitj = (ptitj * ptjti) / ((ptitj + ptjti) - (ptitj * ptjti))
        except ZeroDivisionError:
            rtitj = 0
        temp_list.append(rtitj)
    R_matrix.append(temp_list)
```

In [30]:
```
pd.DataFrame(R_matrix,columns = unique_words[:50]).head()
```

Out[30]:

	conducted	rewritten	clamping	rediscovered	signicantly	hawkingunruh	dsds	doefger	counteract
0	0.000000	0.133333	0.0	0.000000	0.000000	0.000000	0.0	0.000000	0.000000
1	0.133333	0.000000	0.0	0.000000	0.043478	0.000000	0.0	0.133333	0.133333
2	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.0	0.000000	0.000000
3	0.000000	0.000000	0.0	0.000000	0.181818	0.000000	0.0	0.000000	0.000000
4	0.000000	0.043478	0.0	0.181818	0.000000	0.181818	0.0	0.000000	0.000000

5 rows × 50 columns

B. SEMANTIC GRAPH CONSTRUCTION

In [32]:
```
G_adj_matrix = []
W=0.02
```

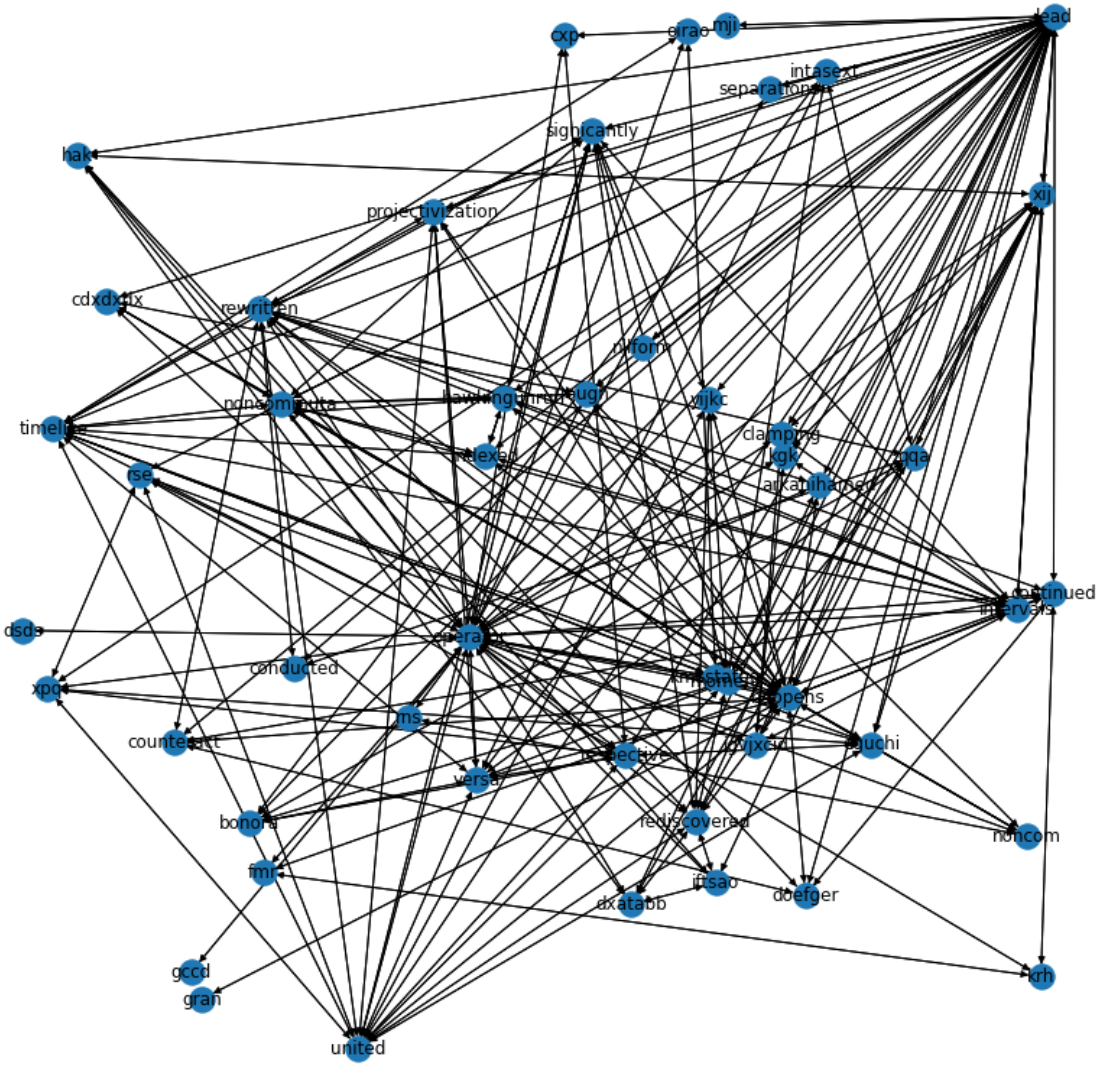
In [33]:
```
for i in range(len(R_matrix)):
    temp_list=[]
    for j in range(len(R_matrix)):
        if R_matrix[i][j]>W:
            temp_list.append(1)
        else:
            temp_list.append(0)
    G_adj_matrix.append(temp_list)
```

In [34]:
```
G = nx.DiGraph(pd.DataFrame(G_adj_matrix).values)
```

In [35]:
```
node_labels = {}
for i in range(50):
    node_labels[i]=unique_words[i]
```

In [36]:
```
plt.figure(3,figsize=(12,12))
nx.draw_random(G,labels = node_labels)
```

```
plt.show()
```



Max frequency Word

In [32]:
```
from collections import defaultdict
freq=defaultdict(int)

for i in unique_words:
    for j in feature_terms:
        for k in j:
            if i==k:
                freq[i]+=1
```

In [33]:
```
max_freq=0
max_freq_word=""
for item in freq.items():
```

```
    if item[1]>max_freq:
        max_freq = item[1]
        max_freq_word = item[0]
```

In [34]:
```
max_freq_word
```

Out[34]:
```
'theory'
```

C. SEMANTIC GRAPH REFINEMENT

1. GENERATING THE CONTEXTUAL RELATIONSHIPS

In [37]:
```
model = gensim.models.Word2Vec(feature_terms, min_count = 1, size = 100,window = 5, sg
```

In [38]:
```
C_matrix=[]
for i in range(50):
    temp_list = []
    for j in range(50):
        if i==j:
            temp_list.append(0)
        else:
            temp_list.append(model.wv.similarity(unique_words[i],unique_words[j]))
    C_matrix.append(temp_list)
```

In [39]:
```
pd.DataFrame(C_matrix).head()
```

Out[39]:

	0	1	2	3	4	5	6	7	8	9	...
0	0.000000	0.852728	0.966205	0.938955	0.875227	0.956123	0.977885	0.868652	0.985750	0.813436	...
1	0.852728	0.000000	0.842137	0.721419	0.652589	0.784927	0.844990	0.598303	0.853692	0.864076	...
2	0.966205	0.842137	0.000000	0.915791	0.862674	0.963949	0.984360	0.804524	0.974618	0.838573	...
3	0.938955	0.721419	0.915791	0.000000	0.942720	0.963700	0.933268	0.847911	0.943481	0.786183	...
4	0.875227	0.652589	0.862674	0.942720	0.000000	0.899552	0.861495	0.760008	0.874705	0.762156	...

5 rows × 50 columns

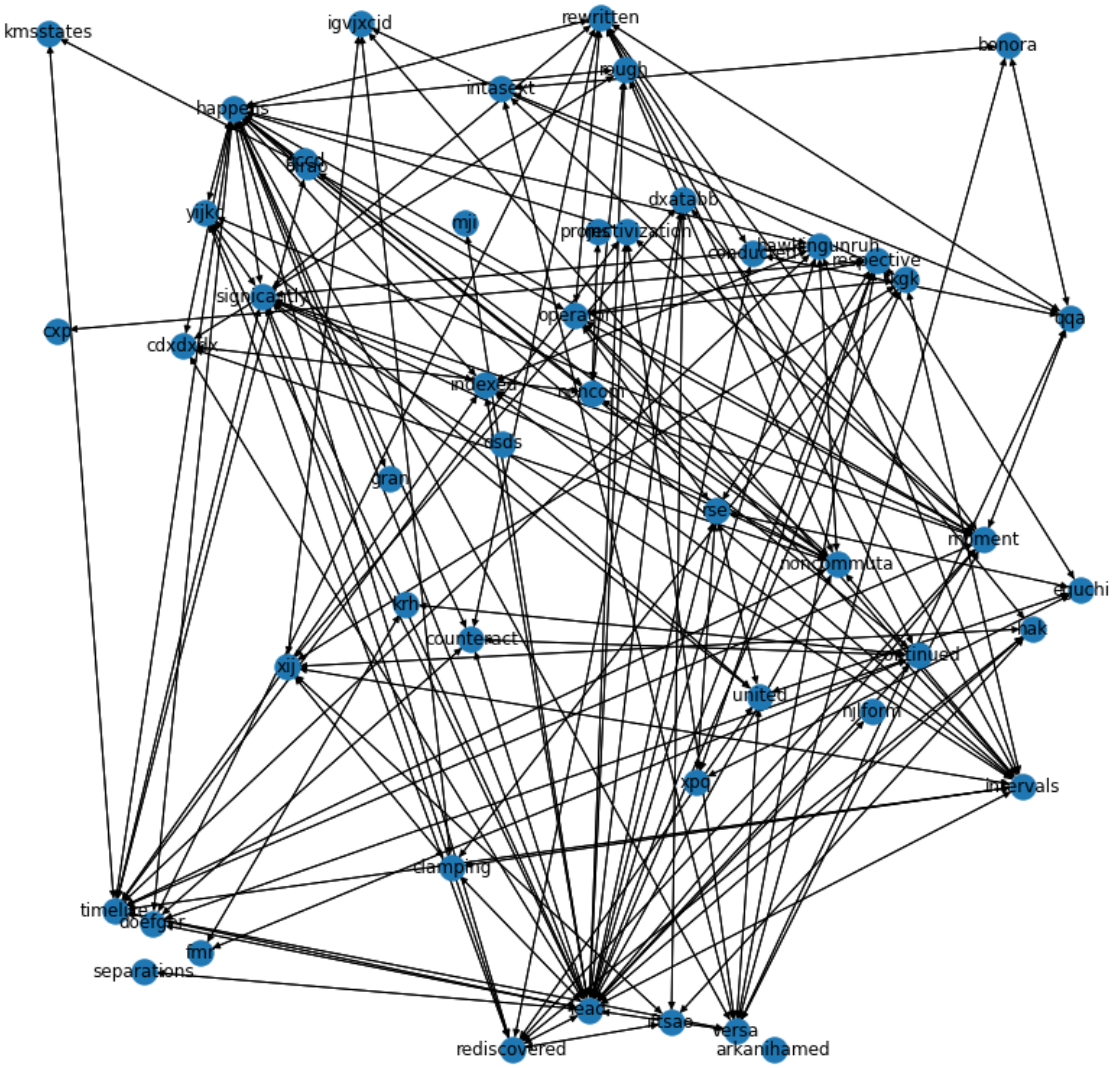
2. REFINING THE SEMANTIC GRAPH

In [40]:
```
k = 0.5

for i in range(len(R_matrix)):
    for j in range(len(R_matrix)):
        if C_matrix[i][j]<k:
            R_matrix[i][j]=0
            G_adj_matrix[i][j]=0
```

In [41]:
```
G = nx.DiGraph(pd.DataFrame(G_adj_matrix).values)
```

In [42]:
```
plt.figure(3,figsize=(12,12))
nx.draw_random(G,labels = node_labels)
plt.show()
```



TOPIC MINING FROM THE REFINED SEMANTIC GRAPH

A. SUBGRAPH SEGMENTATION

In [43]:
```
W_matrix=[]
```

In [44]:
```
for i in range(len(R_matrix)):
    temp_list=[]
```

```
    for j in range(len(R_matrix)):
        temp_list.append(0)
    W_matrix.append(temp_list)

for i in range(len(R_matrix)):
    for j in range(len(R_matrix)):
        if W_matrix[i][j]!=0:
            continue
        W_matrix[i][j]=R_matrix[i][j]
        W_matrix[j][i]=R_matrix[i][j]
```

In [45]:
```
D_matrix=[]
```

In [46]:
```
row_sum=[]
for i in range(len(W_matrix)):
    cur_sum=0
    for j in range(len(W_matrix)):
        cur_sum=cur_sum + W_matrix[i][j]
    row_sum.append(cur_sum)

for i in range(len(W_matrix)):
    temp_list=[]
    for j in range(len(W_matrix)):
        if i==j:
            temp_list.append(row_sum[i])
        else:
            temp_list.append(0)
    D_matrix.append(temp_list)
```

In [47]:
```
L_matrix=[]
```

In [48]:
```
for i in range(len(W_matrix)):
    temp_list=[]
    for j in range(len(W_matrix)):
        temp_list.append(D_matrix[i][j]-W_matrix[i][j])
    L_matrix.append(temp_list)
```

In [ ]:
```

```

N cut Algorithm

In [50]:
```
import sys
from numpy import array, reshape, shape, matrix, ones, zeros, sqrt, sort, arange
from numpy import nonzero, fromfile, tile, append, prod, double, argsort, sign
from numpy import kron, multiply, divide, abs, reshape, asarray
from scipy import rand
from scipy.sparse import csc_matrix, spdiags
from scipy.sparse.linalg.eigen.arpack import eigsh
from scipy.linalg import norm, svd, LinAlgError
```

In [51]:
```
class SVDError(Exception):
    def __init__(self, value):
        self.value = value

    def __str__(self):
        return repr(self.value)
```

In [52]:
```
def ncut(W, nbEigenValues):

    offset = .5
    maxiterations = 100
    eigsErrorTolerence = 1e-6
    truncMin = 1e-6
    eps = 2.2204e-16

    m = shape(W)[1]

    d = abs(W).sum(0)
    dr = 0.5 * (d - W.sum(0))
    d = d + offset * 2
    dr = dr + offset

    W = W + spdiags(dr, [0], m, m, "csc")
    Dinvsqrt = spdiags((1.0 / sqrt(d + eps)), [0], m, m, "csc")
    P = Dinvsqrt * (W * Dinvsqrt);


    eigen_val, eigen_vec = eigsh(P, nbEigenValues, tol=eigsErrorTolerence, which='LA')

    i = argsort(-eigen_val)
    eigen_val = eigen_val[i]
    eigen_vec = eigen_vec[:, i]

    eigen_vec = Dinvsqrt * matrix(eigen_vec)
    norm_ones = norm(ones((m, 1)))
    for i in range(0, shape(eigen_vec)[1]):
        eigen_vec[:, i] = (eigen_vec[:, i] / norm(eigen_vec[:, i])) * norm_ones
        if eigen_vec[0, i] != 0:
            eigen_vec[:, i] = -1 * eigen_vec[:, i] * sign(eigen_vec[0, i])

    return (eigen_val, eigen_vec)
```

In [53]:
```
def discretisation(eigen_vec):
    eps = 2.2204e-16

    [n, k] = shape(eigen_vec)
    vm = kron(ones((1, k)), sqrt(multiply(eigen_vec, eigen_vec).sum(1)))
    eigen_vec = divide(eigen_vec, vm)

    svd_restarts = 0
    exitLoop = 0

    while (svd_restarts < 30) and (exitLoop == 0):

        c = zeros((n, 1))
        R = matrix(zeros((k, k)))
        R[:, 0] = eigen_vec[int(rand(1) * (n)), :].transpose()
```

```
        for j in range(1, k):
            c = c + abs(eigen_vec * R[:, j - 1])
            R[:, j] = eigen_vec[c.argmin(), :].transpose()

        lastObjectiveValue = 0
        nbIterationsDiscretisation = 0
        nbIterationsDiscretisationMax = 20

        while exitLoop == 0:
            nbIterationsDiscretisation = nbIterationsDiscretisation + 1

            tDiscrete = eigen_vec * R

            j = reshape(asarray(tDiscrete.argmax(1)), n)
            eigenvec_discrete = csc_matrix((ones(len(j)), (range(0, n), array(j))), sha


            tSVD = eigenvec_discrete.transpose() * eigen_vec

            try:
                U, S, Vh = svd(tSVD)
                svd_restarts += 1
            except LinAlgError:

                print >> sys.stderr, "SVD did not converge, randomizing and trying agai
                break

            NcutValue = 2 * (n - S.sum())
            if ((abs(NcutValue - lastObjectiveValue) < eps ) or
                    ( nbIterationsDiscretisation > nbIterationsDiscretisationMax )):
                exitLoop = 1
            else:

                lastObjectiveValue = NcutValue
                R = matrix(Vh).transpose() * matrix(U).transpose()

    if exitLoop == 0:
        raise SVDError("SVD did not converge after 30 retries")
    else:
        return (eigenvec_discrete)
```

In [57]:
```
eigen_val, eigen_vec = ncut(np.array(W_matrix),10)
```

In [58]:
```
eigenvec_discrete = discretisation(eigen_vec)
```

```
C:\Users\karthikn\AppData\Local\Temp/ipykernel_17432/480470656.py:15: DeprecationWarnin
g: scipy.rand is deprecated and will be removed in SciPy 2.0.0, use numpy.random.rand in
stead
  R[:, 0] = eigen_vec[int(rand(1) * (n)), :].transpose()
```

In [59]:
```
eigenvec_discrete
```

Out[59]:
```
<50x10 sparse matrix of type '<class 'numpy.float64'>'
        with 50 stored elements in Compressed Sparse Column format>
```

In [60]:
```
eigen_val
```

Out[60]:
```
array([0.83050915, 0.78146289, 0.76148696, 0.74343769, 0.72067215,
       0.68567973, 0.62544683, 0.58751072, 0.57772285, 0.5       ])
```

Topic Clustering

In [61]:
```
from sklearn.mixture import GaussianMixture
```

In [63]:
```
gm = GaussianMixture(n_components=10, random_state=123, n_init=10)
preds = gm.fit_predict(W_matrix)
```

In [64]:
```
preds
```

Out[64]:
```
array([0, 0, 5, 5, 8, 8, 2, 0, 0, 8, 2, 2, 2, 2, 2, 2, 8, 2, 2, 7, 2, 9,
       2, 4, 2, 0, 2, 5, 6, 0, 0, 2, 5, 5, 2, 0, 2, 8, 2, 8, 0, 8, 1, 2,
       0, 0, 8, 3, 2, 0], dtype=int64)
```

In [71]:
```
from collections import defaultdict
clusters=defaultdict(list)
```

In [73]:
```
i = 0
for cluster_no in preds:
    clusters[cluster_no].append(unique_words[i])
    i=i+1
```

In [74]:
```
clusters
```

Out[74]:
```
defaultdict(list,
            {0: ['conducted',
              'rewritten',
              'doefger',
              'counteract',
              'fmr',
              'krh',
              'versa',
              'qqa',
              'bonora',
              'continued',
              'intasext',
              'projectivization'],
             5: ['clamping', 'rediscovered', 'xij', 'hak', 'iftsao'],
             8: ['signicantly',
              'hawkingunruh',
              'moment',
              'noncommuta',
              'happens',
              'lead',
              'intervals',
              'timelike'],
             2: ['dsds',
              'separations',
```

```
 'yijkc',
 'njlform',
 'respective',
 'gran',
 'xpq',
 'oirao',
 'united',
 'arkanihamed',
 'operator',
 'noncom',
 'cxp',
 'eguchi',
 'rns',
 'mji',
 'gccd',
 'kmsstates',
 'rse'],
7: ['dxatabb'],
9: ['rough'],
4: ['kgk'],
6: ['cdxdxdx'],
1: ['igvjxcid'],
3: ['indexed']})
```

In [ ]: