japan_95

March 19, 2020

[1]:
```
%load_ext autoreload
%autoreload 2
%matplotlib inline
```

[2]:
```
import pandas as pd
import numpy as np
```

[3]:
```
import sys

sys.path.append("/disk/home/amy/rankability_toolbox")
```

[4]:
```
import pyrankability
```

0.1 Analysis of the 1995 Japan with trimming

[5]:
```
DATAFILE="/disk/home/amy/data/jpn2005_liot_w_trim/jpn1995_liot.csv"
```

[6]:
```
SECTORSFILE="/disk/home/amy/data/jpn2005_liot_sector_name.csv"
```

[7]:
```
sectors = pd.read_csv(SECTORSFILE,header=None)
sectors = sectors[0]
sectors
```

[7]:
```
0                                        Crop cultivation
1                                               Livestock
2                                  Agricultural services
3                                              Forestry
4                                              Fisheries
                                 …
97                                        Accommodations
98     Cleaning, barber shops, beauty shops and publi…
99                               Other personal services
100                                     Office supplies
101                  Activities not elsewhere classified
Name: 0, Length: 102, dtype: object
```

[8]:
```
D = pd.read_table(DATAFILE,sep=',',header=None)
```

[9]:
```
D
```

[9]:

```
          0         1         2         3         4    5         6    \
0    0.023481  0.104087  0.000000  0.000000  0.000000  0.0  0.000000
1    0.012583  0.107123  0.000000  0.000000  0.000000  0.0  0.000000
2    0.069588  0.046472  0.000000  0.000000  0.000000  0.0  0.000000
3    0.000000  0.000000  0.000000  0.234211  0.000000  0.0  0.000000
4    0.000000  0.000000  0.000000  0.000000  0.049383  0.0  0.000000
..        …         …         …         …         …    …         …
97   0.000000  0.000000  0.000000  0.000000  0.000000  0.0  0.000000
98   0.000000  0.000000  0.000000  0.000000  0.000000  0.0  0.000000
99   0.000000  0.000000  0.000000  0.000000  0.000000  0.0  0.000000
100  0.000000  0.000000  0.000000  0.000000  0.000000  0.0  0.000000
101  0.020168  0.010237  0.039641  0.017655  0.019263  0.0  0.017799

          7         8         9    …  92   93   94   95        96   \
0    0.000000  0.137335  0.054858  …  0.0  0.0  0.0  0.0  0.017944
1    0.000000  0.092838  0.000000  …  0.0  0.0  0.0  0.0  0.000000
2    0.000000  0.000000  0.000000  …  0.0  0.0  0.0  0.0  0.000000
3    0.000000  0.000000  0.000000  …  0.0  0.0  0.0  0.0  0.000000
4    0.000000  0.052294  0.000000  …  0.0  0.0  0.0  0.0  0.000000
..        …         …         …  …  …  …  …  …  …        …
97   0.000000  0.000000  0.000000  …  0.0  0.0  0.0  0.0  0.000000
98   0.000000  0.000000  0.000000  …  0.0  0.0  0.0  0.0  0.000000
99   0.000000  0.000000  0.000000  …  0.0  0.0  0.0  0.0  0.000000
100  0.000000  0.000000  0.000000  …  0.0  0.0  0.0  0.0  0.000000
101  0.018245  0.016881  0.013235  …  0.0  0.0  0.0  0.0  0.000000

          97        98   99   100  101
0    0.013755  0.000000  0.0  0.0  0.0
1    0.000000  0.000000  0.0  0.0  0.0
2    0.000000  0.000000  0.0  0.0  0.0
3    0.000000  0.000000  0.0  0.0  0.0
4    0.000000  0.000000  0.0  0.0  0.0
..        …         …   …  …  …  …
97   0.000000  0.000000  0.0  0.0  0.0
98   0.000000  0.028682  0.0  0.0  0.0
99   0.000000  0.000000  0.0  0.0  0.0
100  0.000000  0.000000  0.0  0.0  0.0
101  0.000000  0.000000  0.0  0.0  0.0

[102 rows x 102 columns]
```

[10]:
```
D.columns=sectors
```

[11]:
```
D.index=sectors
```

[12]:
```
D.isnull().sum()
```

[12]:
```
0
Crop cultivation                                            0
Livestock                                                   0
Agricultural services                                       0
Forestry                                                    0
Fisheries                                                   0
                                                           ..
Accommodations                                              0
Cleaning, barber shops, beauty shops and public baths       0
Other personal services                                     0
Office supplies                                             0
Activities not elsewhere classified                         0
Length: 102, dtype: int64
```

[13]:
```
D.dtypes
```

[13]:
```
0
Crop cultivation                                            float64
Livestock                                                   float64
Agricultural services                                       float64
Forestry                                                    float64
Fisheries                                                   float64
                                                             …
Accommodations                                              float64
Cleaning, barber shops, beauty shops and public baths       float64
Other personal services                                     float64
Office supplies                                             float64
Activities not elsewhere classified                         float64
Length: 102, dtype: object
```

0.2 Run the lp solver

[14]:
```
k,details = pyrankability.lop.lp(D.values)
```

```
Academic license - for non-commercial use only
```

[98]:
```
k
```

[98]:
```
46.85598382000097
```

0.3 Run the bilp solver and report up to 50 unique solutions

[ ]:
```
k_bilp,details_bilp = pyrankability.lop.bilp(D.
↪values,max_solutions=1,num_random_restarts=50)
```

[99]:
```
k_bilp
```

[99]:
```
46.855983820000006
```

[100]:
```
len(details_bilp['P'])
```

[100]:
```
51
```

0.4 What is saved in the details

In case we want, I have stored the first P found, all the P found, the first x found, the objective values each time, and the x matrices each time

[101]:
```
details_bilp.keys()
```

[101]:
```
dict_keys(['Pfirst', 'P', 'x', 'objs', 'xs'])
```

0.5 Important to check

It is important to check and make sure all solutions are optimal. Running this check helped me sort out a bug or two.

[102]:
```
objs=[]
for perm in details_bilp['P']:
    objs.append(pyrankability.lop.objective_count_perm(D.values,perm))
```

[104]:
```
pd.Series(objs)
```

[104]:
```
0     46.855984
1     46.855984
2     46.855984
3     46.855984
4     46.855984
5     46.855984
6     46.855984
7     46.855984
8     46.855984
9     46.855984
10    46.855984
11    46.855984
```

```
12    46.855984
13    46.855984
14    46.855984
15    46.855984
16    46.855984
17    46.855984
18    46.855984
19    46.855984
20    46.855984
21    46.855984
22    46.855984
23    46.855984
24    46.855984
25    46.855984
26    46.855984
27    46.855984
28    46.855984
29    46.855984
30    46.855984
31    46.855984
32    46.855984
33    46.855984
34    46.855984
35    46.855984
36    46.855984
37    46.855984
38    46.855984
39    46.855984
40    46.855984
41    46.855984
42    46.855984
43    46.855984
44    46.855984
45    46.855984
46    46.855984
47    46.855984
48    46.855984
49    46.855984
50    46.855984
dtype: float64
```

0.6 Now let's put the unique solutions in a Pandas dataframe

[105]:
```
P_unique = list(set(details_bilp['P']))
P_df = pd.DataFrame(P_unique)
P_df
```

[105]:

	0	1	2	3	4	5	6	7	8	9	…	92	93	94	95	\
0	94	72	73	71	93	88	92	7	27	67	…	9	86	96	66	
1	94	72	73	71	93	88	92	7	27	67	…	52	56	61	49	
2	94	72	73	71	93	88	92	7	27	67	…	97	47	100	52	
3	94	72	73	71	93	92	88	7	27	67	…	53	52	47	100	
4	94	72	73	71	93	88	92	7	27	67	…	78	68	89	56	
5	94	72	73	71	93	92	88	7	27	67	…	74	99	53	47	
6	94	72	73	71	93	92	88	7	27	67	…	69	53	45	79	
7	94	72	73	71	93	88	92	7	27	67	…	9	31	96	47	
8	94	72	73	71	93	88	92	7	27	67	…	86	100	66	49	
9	94	72	73	71	93	88	92	7	27	67	…	69	70	86	96	
10	94	72	73	71	93	88	92	7	27	67	…	65	98	89	47	
11	94	72	73	71	93	92	88	7	27	67	…	79	52	56	45	
12	94	72	73	71	93	92	88	7	27	67	…	78	75	57	49	
13	94	72	73	71	93	92	88	7	27	67	…	75	70	86	57	
14	94	72	73	71	93	92	88	7	27	67	…	78	75	70	97	
15	94	72	73	71	93	92	88	7	27	67	…	86	9	53	31	
16	94	72	73	71	93	92	88	7	27	67	…	98	89	100	52	
17	94	72	73	71	93	88	92	7	27	67	…	57	61	31	87	
18	94	72	73	71	93	92	88	7	27	67	…	75	70	99	9	
19	94	72	73	71	93	92	88	7	27	67	…	96	65	56	75	
20	94	72	73	71	93	88	92	7	27	67	…	89	87	78	49	
21	94	72	73	71	93	88	92	7	27	67	…	11	70	86	79	
22	94	72	73	71	93	88	92	7	27	67	…	89	49	9	97	
23	94	72	73	71	93	88	92	7	27	67	…	69	9	97	96	
24	94	72	73	71	93	88	92	7	27	67	…	86	52	99	56	
25	94	72	73	71	93	92	88	7	27	67	…	53	78	74	98	
26	94	72	73	71	93	88	92	7	27	67	…	51	78	11	74	
27	94	72	73	71	93	88	92	7	27	67	…	53	69	96	98	
28	94	72	73	71	93	92	88	7	27	67	…	99	11	56	61	
29	94	72	73	71	93	92	88	7	27	67	…	13	90	69	98	
30	94	72	73	71	93	92	88	7	27	67	…	13	90	69	98	
31	94	72	73	71	93	88	92	7	27	67	…	96	53	86	100	
32	94	72	73	71	93	88	92	7	27	67	…	61	56	98	89	
33	94	72	73	71	93	92	88	7	27	67	…	70	86	97	51	
34	94	72	73	71	93	92	88	7	27	67	…	84	69	98	89	
35	94	72	73	71	93	92	88	7	27	67	…	89	49	96	66	
36	94	72	73	71	93	88	92	7	27	67	…	51	11	86	63	
37	94	72	73	71	93	92	88	7	27	67	…	47	100	68	78	
38	94	72	73	71	93	92	88	7	27	67	…	60	89	75	78	
39	94	72	73	71	93	92	88	7	27	67	…	49	57	52	98	

40	94	72	73	71	93	92	88	7	27	67	…	66	96	56	98
41	94	72	73	71	93	92	88	7	27	67	…	99	9	96	11
42	94	72	73	71	93	88	92	7	27	67	…	86	47	57	100
43	94	72	73	71	93	88	92	7	27	67	…	86	66	87	52
44	94	72	73	71	93	92	88	7	27	67	…	66	61	86	100
45	94	72	73	71	93	92	88	7	27	67	…	66	56	87	65
46	94	72	73	71	93	92	88	7	27	67	…	47	97	96	100
47	94	72	73	71	93	88	92	7	27	67	…	60	75	70	86
48	94	72	73	71	93	92	88	7	27	67	…	66	74	87	61
49	94	72	73	71	93	88	92	7	27	67	…	86	65	63	47
50	94	72	73	71	93	88	92	7	27	67	…	52	66	65	51

	96	97	98	99	100	101
0	61	97	89	31	51	68
1	51	97	45	66	99	31
2	79	56	96	51	11	89
3	56	51	49	45	31	11
4	96	75	70	86	97	49
5	100	66	68	56	78	97
6	96	74	98	89	100	97
7	97	74	11	100	49	45
8	56	65	11	57	53	96
9	98	89	31	100	97	74
10	86	100	52	56	97	96
11	57	75	70	97	86	99
12	9	96	70	97	86	53
13	89	31	56	47	97	100
14	86	96	51	56	99	66
15	96	97	52	56	100	65
16	56	86	11	45	61	53
17	68	75	70	100	97	86
18	97	96	86	63	87	65
19	70	86	63	68	99	97
20	96	57	61	100	66	74
21	9	97	74	96	100	89
22	96	45	66	52	56	31
23	98	89	53	63	78	66
24	78	66	79	87	68	11
25	31	66	9	89	96	97
26	47	100	68	79	53	49
27	11	86	74	97	51	89
28	51	78	97	45	87	74
29	9	96	57	89	86	97
30	53	89	56	9	96	97
31	51	56	79	61	97	57
32	97	96	53	45	65	51
33	99	68	65	66	52	56

```
34   96   53  100   66   97   56
35   74   57   47  100   51   78
36   96   89   49   56   61   79
37   11   96   57   66   79   45
38    9   96   61   70   86   97
39   89   63   11   96   56   45
40   31   89  100   79   61   68
41   60   78   75   70   97   86
42   97   61   68   52   56   89
43   56   99   89   61   65   31
44   11   31   96   87   99   78
45   45   47  100   53   63   68
46   66   53   98   89   56   45
47   78   97   87   52   56   63
48   89   68   52   65   56   53
49  100   96   89   31   11   68
50   31   99   45   49   89   56

[51 rows x 102 columns]
```

[117]:
```
P_series=P_df.T.stack()
```

[124]:
```
pos_df = P_series.index.to_frame()
pos_df.columns=["Position","Solution Index"]
pos_df["Item"] = P_series
pos_df.set_index("Solution Index",inplace=True)
pos_df
```

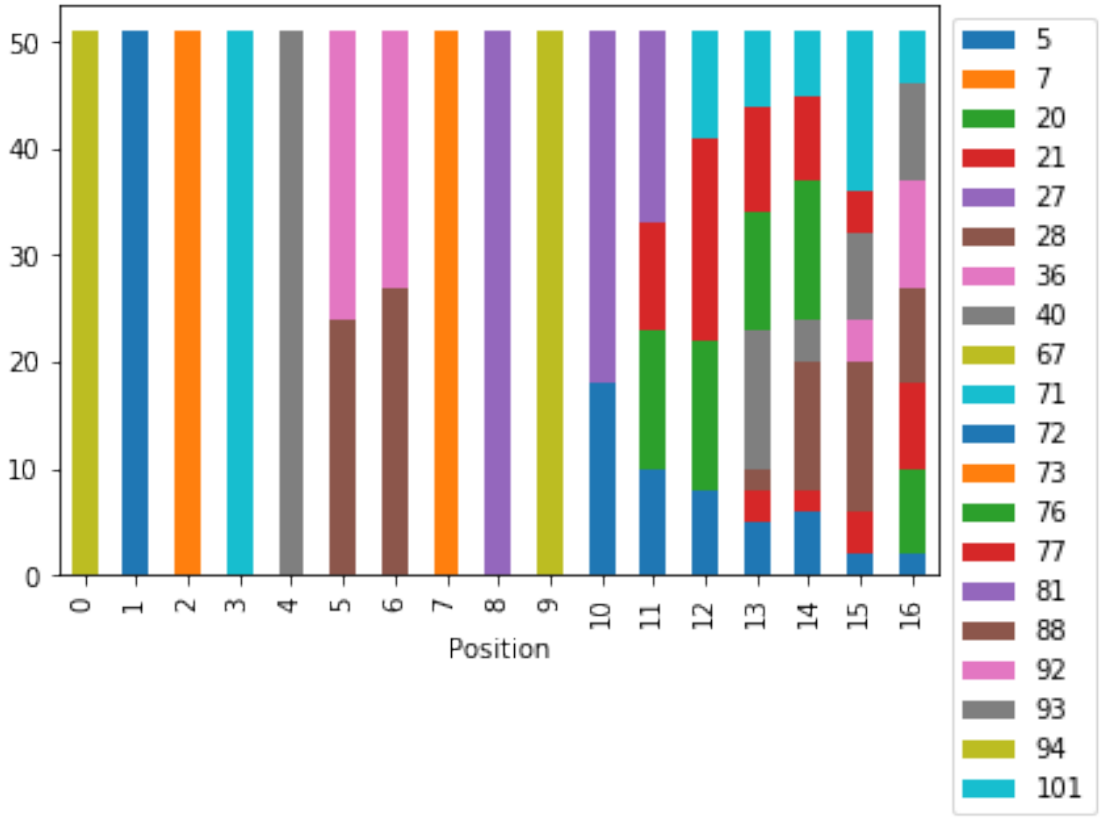
[124]:
```
                Position  Item
Solution Index
0                      0    94
1                      0    94
2                      0    94
3                      0    94
4                      0    94
…                    …   …
46                   101    45
47                   101    63
48                   101    53
49                   101    68
50                   101    56

[5202 rows x 2 columns]
```

0.7 Visualizing some of the top items

In the graph below the x axis is the rank or position in the solutions. I'm keeping it simple and showing the top 16. What I would point out is that things like items 77 and 21 (unfortunately they are both red, so I need more colors). I think that constitutes a lot of movement. If I've got everything aligned correct 77 and 21 ware associated with Water transport and Petrochemical basic products. Item 5 is another interesting case that goes from position 10 to position 16. Item 5 is Metallic ores. So the big question is so what? Is this the breakthrough reason for our approach? Would anyone be interested in these alternative solutions?

[164]:
```
graph_df = pos_df.groupby("Position")["Item"].value_counts().to_frame()
graph_df.columns = ["Count"]
graph_df = graph_df.head(50).reset_index().set_index("Position").
 ↪pivot(columns="Item").fillna(0)
graph_df.columns = [v[1] for v in graph_df.columns]
graph_df.plot.bar(stacked=True).legend(bbox_to_anchor=(1, 1))
```

[164]:
```
<matplotlib.legend.Legend at 0x7f4d97120d68>
```



[168]:
```
D.index[77],D.index[101],D.index[21],D.index[5]
```

[168]:
```
('Water transport',
 'Activities not elsewhere classified',
 'Petrochemical basic products',
 'Metallic ores')
```

0.8 Save the results for later

[106]:
```
#!mkdir /disk/home/amy/results/
```

[107]:
```
import joblib
joblib.dump(details_bilp,"/disk/home/amy/results/japan_95_details_bilp.joblib.
 ↪z")
```

[107]:
```
['/disk/home/amy/results/japan_95_details_bilp.joblib.z']
```

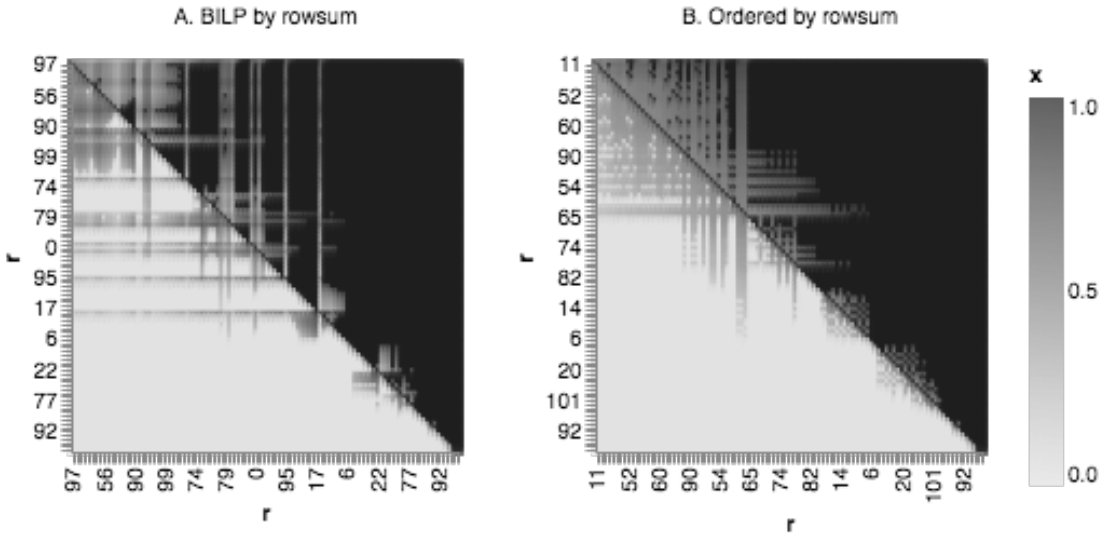
0.9 Below is analysis of the lp results

[111]:
```
Xstar = pyrankability.lop.
 ↪threshold_x(details['x'],lower_cut=1e-3,upper_cut=1-1e-3)

record_ixs = []
r = np.sum(Xstar,axis=1)
ixs = np.argsort(r)
record_ixs.append(ixs)
Xstar_rowsum = Xstar[np.ix_(ixs,ixs)]

r = np.sum(details_bilp['x'],axis=1)
ixs = np.argsort(r)
record_ixs.append(ixs)
Xstar_bilp = Xstar[np.ix_(ixs,ixs)]

labels = ["A. BILP by rowsum", "B. Ordered by rowsum"]
Xstars = [Xstar_bilp,Xstar_rowsum]
```

[112]:
```
import altair as alt
alt.data_transformers.disable_max_rows()

all_df = pd.DataFrame(columns=["i","j","x","Label","oi","oj"])
for ix,Xstar in enumerate(Xstars):
    x = pd.DataFrame(Xstar)
    df = x.stack().reset_index()
    df.columns=["i","j","x"]
    df["oi"] = df["i"].copy()
    df["oj"] = df["j"].copy()
```

```
    df["i"] = record_ixs[ix][df["i"]]
    df["j"] = record_ixs[ix][df["j"]]
    df["Label"] = labels[ix]
    all_df = all_df.append(df)

#all_df = all_df.loc[(all_df.x != 0) & (all_df.x != 1)]
alt.Chart(all_df).mark_square().encode(
    x=alt.X(
        'i:N',
        axis=alt.Axis(labelOverlap="parity"),
        title="r",
        sort=alt.EncodingSortField(field="oi",order="ascending") # The order to␣
 ↪sort in
    ),
    y=alt.Y(
        'j:N',
        axis=alt.Axis(labelOverlap="parity"),
        title="r",
        sort=alt.EncodingSortField(field="oj",order="ascending") # The order to␣
 ↪sort in
    ),
    color=alt.Color("x",scale=alt.Scale(scheme='greys'))
).properties(
    width=180,
    height=180
).facet(
    column=alt.Column("Label:N", title=None)
).resolve_scale(x='independent',y='independent')
```

[112]:



[114]:
```
pd.Series(D.columns[record_ixs[0]])
```

[114]:
```
0                                        Accommodations
1      Medical service, health, social security and n…
2                              Eating and drinking places
3                                   Public administration
4      Cleaning, barber shops, beauty shops and publi…
                               …
97                  Repair of motor vehicles and machine
98                                              Commerce
99              Real estate agencies and rental services
100                                 Finance and insurance
101                               Other business services
Name: 0, Length: 102, dtype: object
```

0.10 Still in development

This is still in development for this size.

[26]:
```
# P,info = pyrankability.lop.find_P_from_x(D.values,k,details)
```

```
Going to loop for 2.0
Going to loop for 4194304.0
Going to loop for 524288.0
Going to loop for 2.4049076047604052e+111
```

```
      ␣
↪---------------------------------------------------------------------------

       KeyboardInterrupt                         Traceback (most recent call␣
↪last)

       <ipython-input-26-9668fe832127> in <module>
   ----> 1 P,info = pyrankability.lop.find_P_from_x(D.
↪values,k,details,lower_cut=0.4,upper_cut=1-0.4)


       ~/rankability_toolbox/pyrankability/lop.py in find_P_from_x(D, k,␣
↪details, lower_cut, upper_cut)
       203             Xsub = np.round(new_details['x'])
       204             Xsub[np.ix_(group,group)] = Xstar[np.ix_(group,group)]
   --> 205             obj_sub, permutations_sub =␣
↪objective_count(Dordered,Dordered[np.ix_(group,group)],Xsub,k,group)
       206             perm_iters.append((group[0],permutations_sub))
       207     permutations = generate_perms(perm_iters)
```

```
      ~/rankability_toolbox/pyrankability/lop.py in objective_count(Dordered,␣
↪D, Xstar, min_value, group)
      267         return min_value,Xsub_min_value,frac_ixs_min_value
      268
  --> 269     results = Parallel(n_jobs=cpu_count)(delayed(compute)(init_seq)␣
↪for init_seq in init_seqs)
      270     # combine
      271     #min_value = np.Inf


      /data/env/lib/python3.6/site-packages/joblib/parallel.py in␣
↪__call__(self, iterable)
     1014
     1015             with self._backend.retrieval_context():
  -> 1016                 self.retrieve()
     1017             # Make sure that we get a last message telling us we are␣
↪done
     1018             elapsed_time = time.time() - self._start_time


      /data/env/lib/python3.6/site-packages/joblib/parallel.py in␣
↪retrieve(self)
      906             try:
      907                 if getattr(self._backend, 'supports_timeout', False):
  --> 908                     self._output.extend(job.get(timeout=self.
↪timeout))
      909                 else:
      910                     self._output.extend(job.get())


      /data/env/lib/python3.6/site-packages/joblib/_parallel_backends.py in␣
↪wrap_future_result(future, timeout)
      552         AsyncResults.get from multiprocessing."""
      553         try:
  --> 554             return future.result(timeout=timeout)
      555         except LokyTimeoutError:
      556             raise TimeoutError()


      /usr/lib/python3.6/concurrent/futures/_base.py in result(self, timeout)
      425                 return self.__get_result()
      426
  --> 427             self._condition.wait(timeout)
      428
      429             if self._state in [CANCELLED, CANCELLED_AND_NOTIFIED]:
```

```
      /usr/lib/python3.6/threading.py in wait(self, timeout)
      293         try:    # restore state no matter what (e.g.,␣
↪KeyboardInterrupt)
      294             if timeout is None:
  --> 295                 waiter.acquire()
      296                 gotit = True
      297             else:


      KeyboardInterrupt:
```