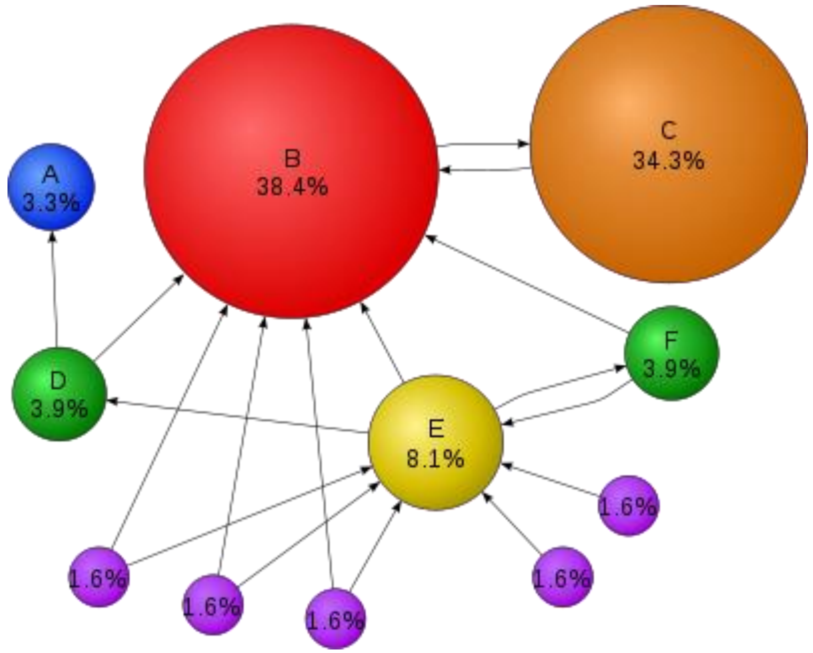
PageRank applied to 2016 TFL Cycle Hire Journey Data

(the secret life of Boris Bikes, Part II)

JOHN DOWNING:IS71059B:ASSIGNMENT 2

PageRank

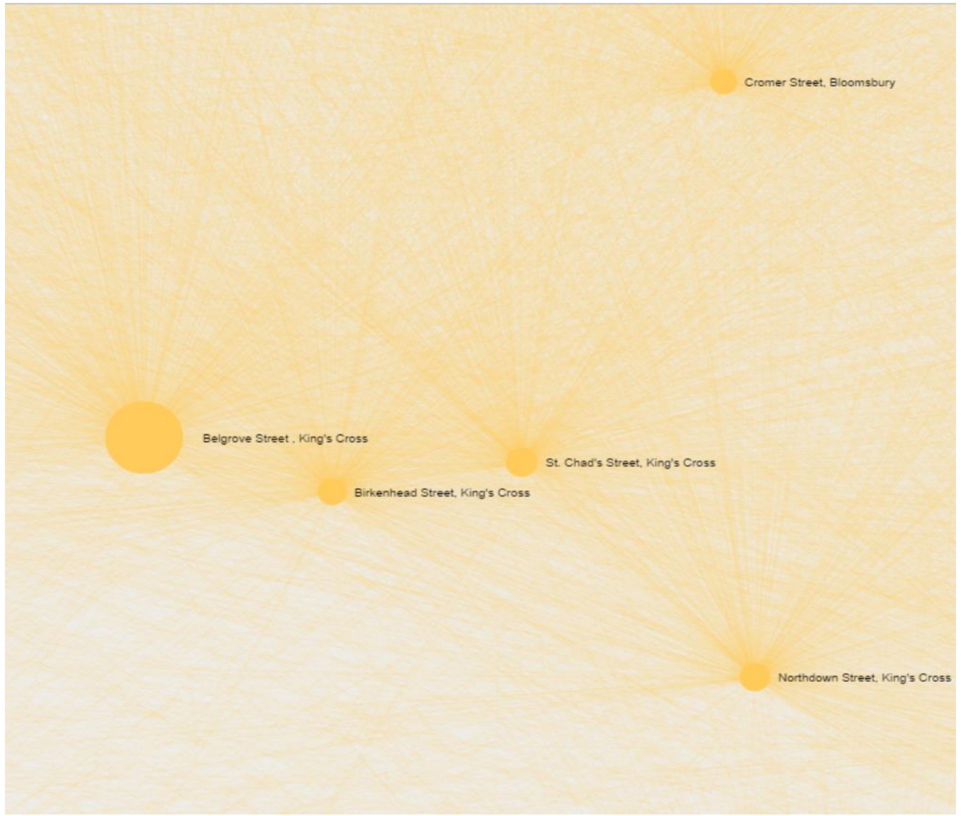
- Made famous by Google (Larry *Page*).
- Ranks web pages in terms of their importance, for ordering of search results.
- Focuses on the hyperlinks which connect web pages.
- Does not rank (directly) on the number of links which point to a web page.
- Ranks on the importance (rank) of the pages containing those links.
- What’s this got to do with Boris Bikes..?



Source: https://en.wikipedia.org/wiki/File:PageRanks-Example.svg

Graph Models

- Designed to model connected data.
- Connections between people, places, things…
- PageRank can be applied generically to any graph, not just web pages.
- Attempts to uncover influential nodes in network.
- Possible to model the TFL journey data as a graph – with docking stations (nodes) connected by journeys (directed edges).

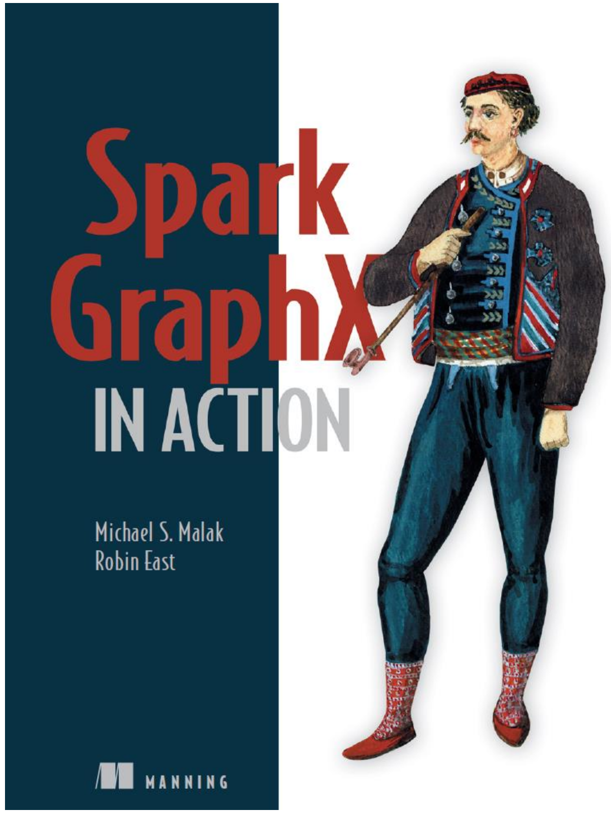


TFL Cycle Hire Journey Data 2016

- 9,018,398 rows representing single journeys between two different docking stations.
- 789 docking stations, 373,814 journey combinations.
- Pre-processed as part of Data Visualisation Assignment 3.
- Previously analysed using Pandas DataFrame.
- Good for simple aggregation: journey counts, busiest docking stations.
- Bad for traversal (e.g. “friends of friends of friends”).
- Would PageRank find any new insights?

GraphX

- Included in Apache Spark.
- Used under the covers by some of the Spark MLlib algorithms.
- Scala API.
- Built-in graph processing algorithms, including PageRank.



Malak, M. S., & East, R. (2016). *Spark GraphX in action*.

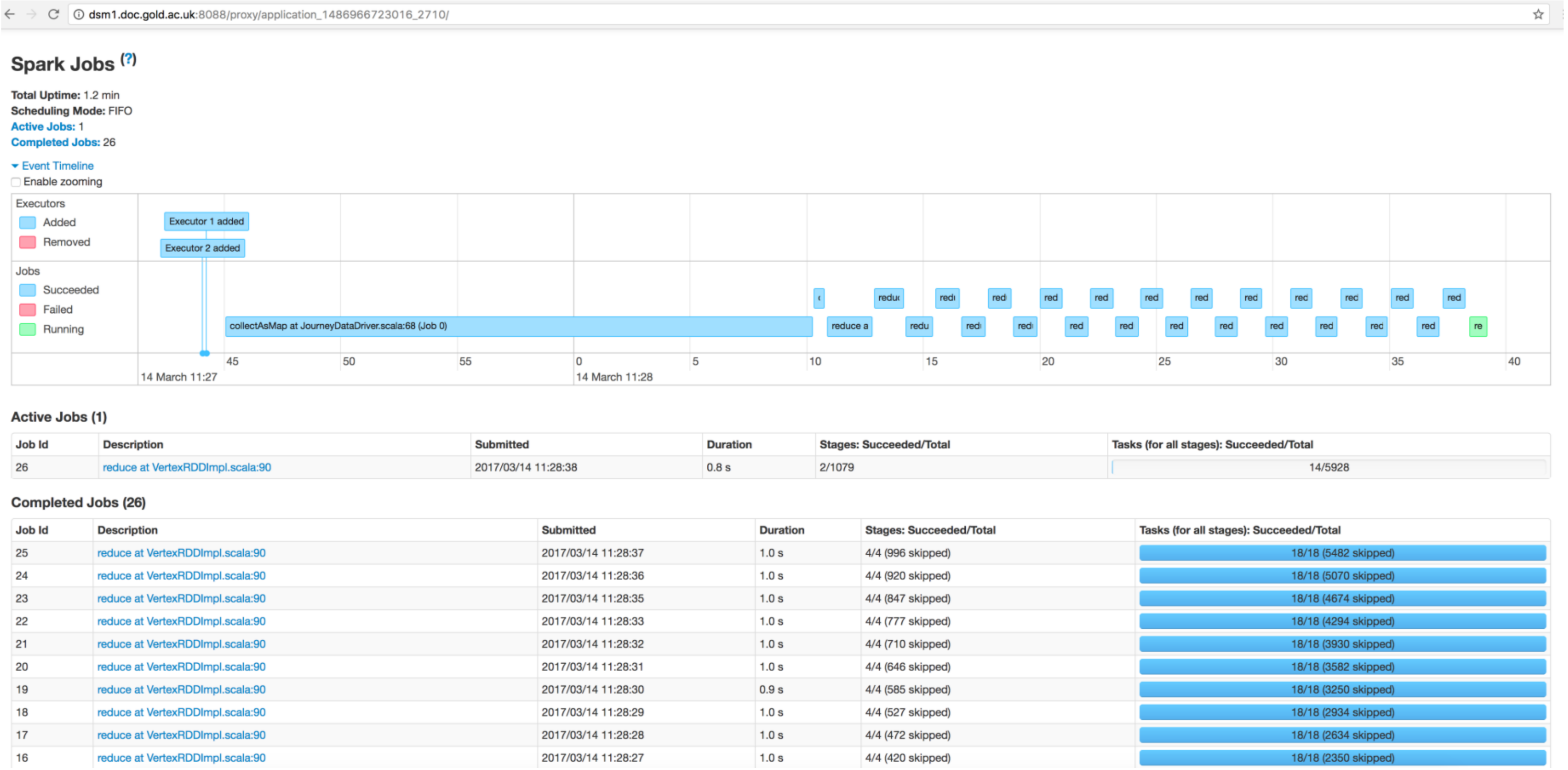
Method

- Compile Scala code into jar file and copy to cluster.
- Load docking stations csv file into Spark, convert to vertices.
- Load journey data csv file into Spark, convert to edges.
- Create Graph RDD and run pageRank algorithm.
- Run personalPageRank on selected vertices.
- Perform SparkSQL queries, output results as csv.
- Write graph data in GEXF format, load into Gephi (desktop app).
- Export from Gephi for use in Javascript visualisations (doc.gold.ac.uk/~jdown003/network/).

Execution

```
spark-submit \
--class com.downinja.msc.bda.JourneyDataDriver \
--master yarn \
--deploy-mode client \
--num-executors 10 \
journey-data-driver-0.0.1-SNAPSHOT.jar \
hdfs:/user/jdown003/ \
/home/jdown003/sparkstuff/ \
RegularJourneys2016.csv
```

Execution (YARN)


dsm1.doc.gold.ac.uk:8088/proxy/application_1486966723016_2710/

Spark Jobs (?)

Total Uptime: 1.2 min
Scheduling Mode: FIFO
Active Jobs: 1
Completed Jobs: 26

Event Timeline
Enable zooming

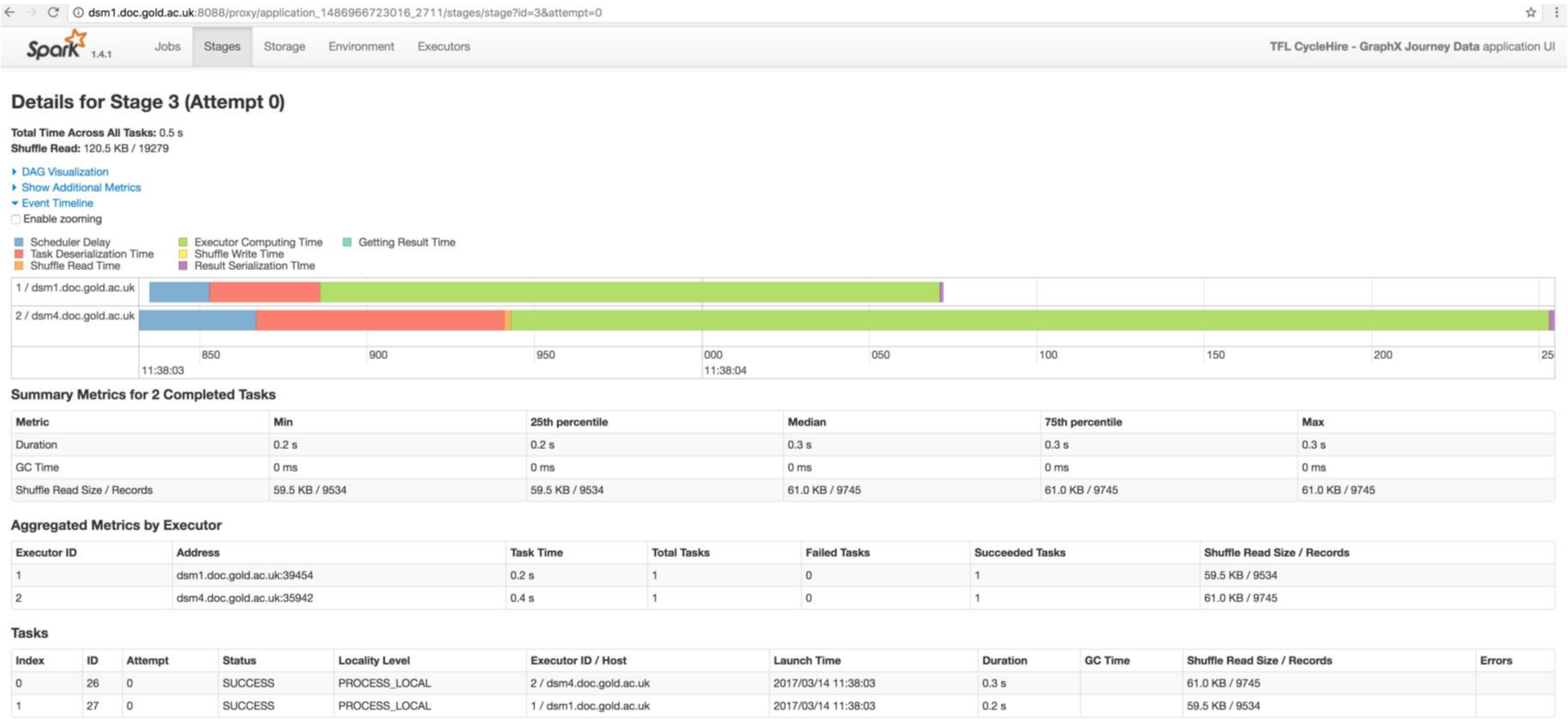
Active Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
26	reduce at VertexRDDImpl.scala:90	2017/03/14 11:28:38	0.8 s	2/1079	14/5928

Completed Jobs (26)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
25	reduce at VertexRDDImpl.scala:90	2017/03/14 11:28:37	1.0 s	4/4 (996 skipped)	18/18 (5482 skipped)
24	reduce at VertexRDDImpl.scala:90	2017/03/14 11:28:36	1.0 s	4/4 (920 skipped)	18/18 (5070 skipped)
23	reduce at VertexRDDImpl.scala:90	2017/03/14 11:28:35	1.0 s	4/4 (847 skipped)	18/18 (4674 skipped)
22	reduce at VertexRDDImpl.scala:90	2017/03/14 11:28:33	1.0 s	4/4 (777 skipped)	18/18 (4294 skipped)
21	reduce at VertexRDDImpl.scala:90	2017/03/14 11:28:32	1.0 s	4/4 (710 skipped)	18/18 (3930 skipped)
20	reduce at VertexRDDImpl.scala:90	2017/03/14 11:28:31	1.0 s	4/4 (646 skipped)	18/18 (3582 skipped)
19	reduce at VertexRDDImpl.scala:90	2017/03/14 11:28:30	0.9 s	4/4 (585 skipped)	18/18 (3250 skipped)
18	reduce at VertexRDDImpl.scala:90	2017/03/14 11:28:29	1.0 s	4/4 (527 skipped)	18/18 (2934 skipped)
17	reduce at VertexRDDImpl.scala:90	2017/03/14 11:28:28	1.0 s	4/4 (472 skipped)	18/18 (2634 skipped)
16	reduce at VertexRDDImpl.scala:90	2017/03/14 11:28:27	1.0 s	4/4 (420 skipped)	18/18 (2350 skipped)

Execution (YARN)



dsm1.doc.gold.ac.uk:8088/proxy/application_1486966723016_2711/stages/stage?id=3&attempt=0

Spark 1.4.1 Jobs Stages Storage Environment Executors

TFL CycleHire - GraphX Journey Data application UI

Details for Stage 3 (Attempt 0)

Total Time Across All Tasks: 0.5 s
Shuffle Read: 120.5 KB / 19279

- DAG Visualization
- Show Additional Metrics
- Event Timeline

Enable zooming

Summary Metrics for 2 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	0.2 s	0.2 s	0.3 s	0.3 s	0.3 s
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Shuffle Read Size / Records	59.5 KB / 9534	59.5 KB / 9534	61.0 KB / 9745	61.0 KB / 9745	61.0 KB / 9745

Aggregated Metrics by Executor

Executor ID	Address	Task Time	Total Tasks	Failed Tasks	Succeeded Tasks	Shuffle Read Size / Records
1	dsm1.doc.gold.ac.uk:39454	0.2 s	1	0	1	59.5 KB / 9534
2	dsm4.doc.gold.ac.uk:35942	0.4 s	1	0	1	61.0 KB / 9745

Tasks

Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Shuffle Read Size / Records	Errors
0	26	0	SUCCESS	PROCESS_LOCAL	2 / dsm4.doc.gold.ac.uk	2017/03/14 11:38:03	0.3 s		61.0 KB / 9745	
1	27	0	SUCCESS	PROCESS_LOCAL	1 / dsm1.doc.gold.ac.uk	2017/03/14 11:38:03	0.2 s		59.5 KB / 9534	

Spark SQL

```
// And in order to summarise these interesting events, we can su
// IN_DEGREE and RANK values and pick out e.g. the vertex with t
// total increase in rank over those vertices which have a large
// IN_DEGREE.
sql =
  "SELECT " +
      "a.ID, " +
      "SUM(b.IN_DEGREE - a.IN_DEGREE) as TOTAL_DEGREE_DIFF, " +
      "SUM(a.RANK - b.RANK) AS TOTAL_RANK_DIFF " +
  "FROM " +
      "VERTICES a, " +
      "VERTICES b " +
  "WHERE " +
      "a.RANK > b.RANK " +
      "AND b.IN_DEGREE > a.IN_DEGREE " +
      "GROUP BY a.ID"

dataFrame = sqlContext.sql(sql)
```

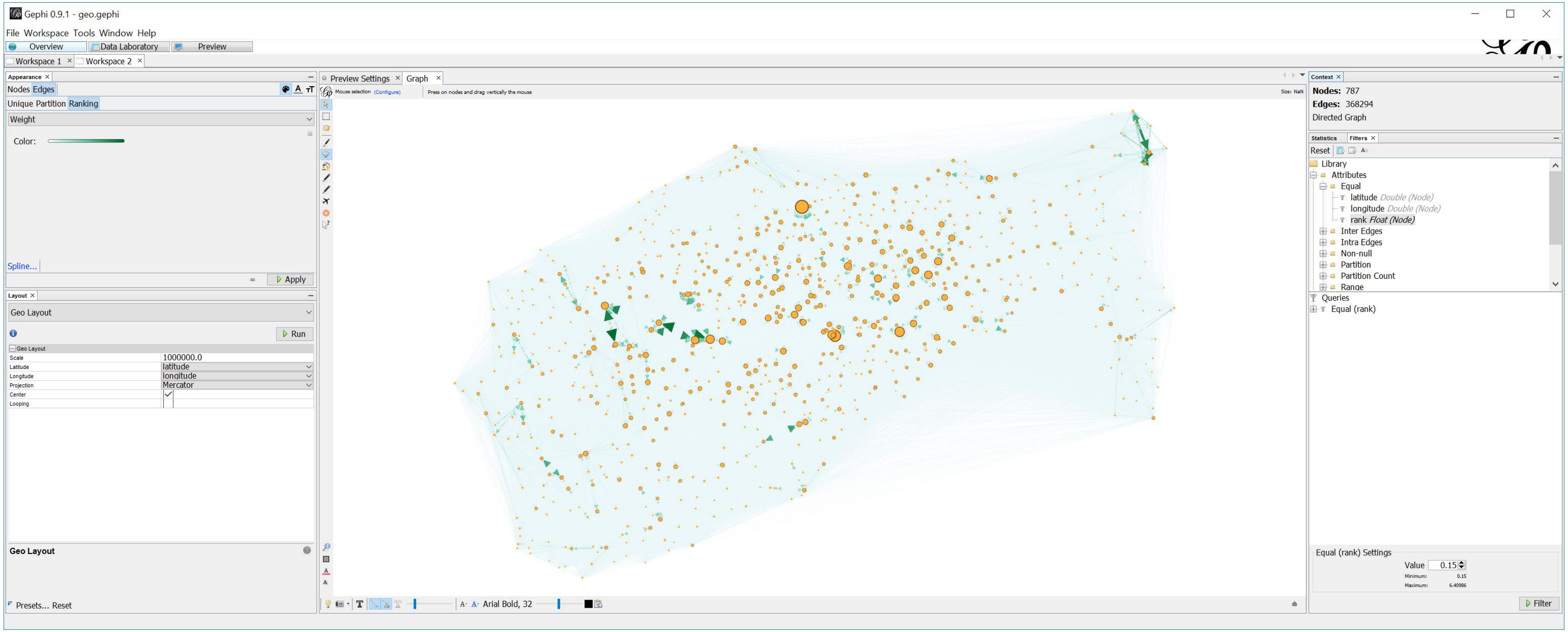
```
ID,TOTAL_DEGREE_DIFF,TOTAL_RANK_DIFF
481,288681,25.67699398
551,310119,23.14778508
532,246537,22.91246599
685,195339,14.99340533
735,156199,13.57869874
621,142202,11.7116798
682,135266,9.990447459
708,155187,9.921007089
730,123372,8.957345516
766,112667,8.091891686
607,99439,8.032583367
570,100687,7.696504958
596,113410,7.546538803
691,87086,7.145611898
707,97804,6.859512114
644,92800,6.726464215
591,83130,6.296761152
671,59093,6.192822674
723,76522,5.681109771
613,55532,5.054251274
547,64778,4.839307311
…
```

GEXF

- XML representation of graph data.
- Open standard.
- Can be read directly by Gephi, SigmaJS, GEXF-JS.

```
<?xml version="1.0" encoding="UTF-8"?>
<gexf xmlns="http://www.gexf.net/1.2draft" version="1.2">
  <graph mode="static" defaultedgetype="directed">
    <attributes class="node">
      <attribute id="0" title="latitude" type="double"/>
      <attribute id="1" title="longitude" type="double"/>
      <attribute id="2" title="rank" type="float"/>
    </attributes>
    <nodes>
      <node id="14" label="Belgrove Street , King's Cross">
        <attvalues>
          <attvalue for="0" value="51.52994371"/>
          <attvalue for="1" value="-0.123616824"/>
          <attvalue for="2" value="6.409864581607833"/>
        </attvalues>
      </node>
      ...
    </nodes>
    <edges>
      <edge source="307" target="404" label="4368" weight="4368.0"/>
      ...
    </edges>
  </graph>
</gexf>
```

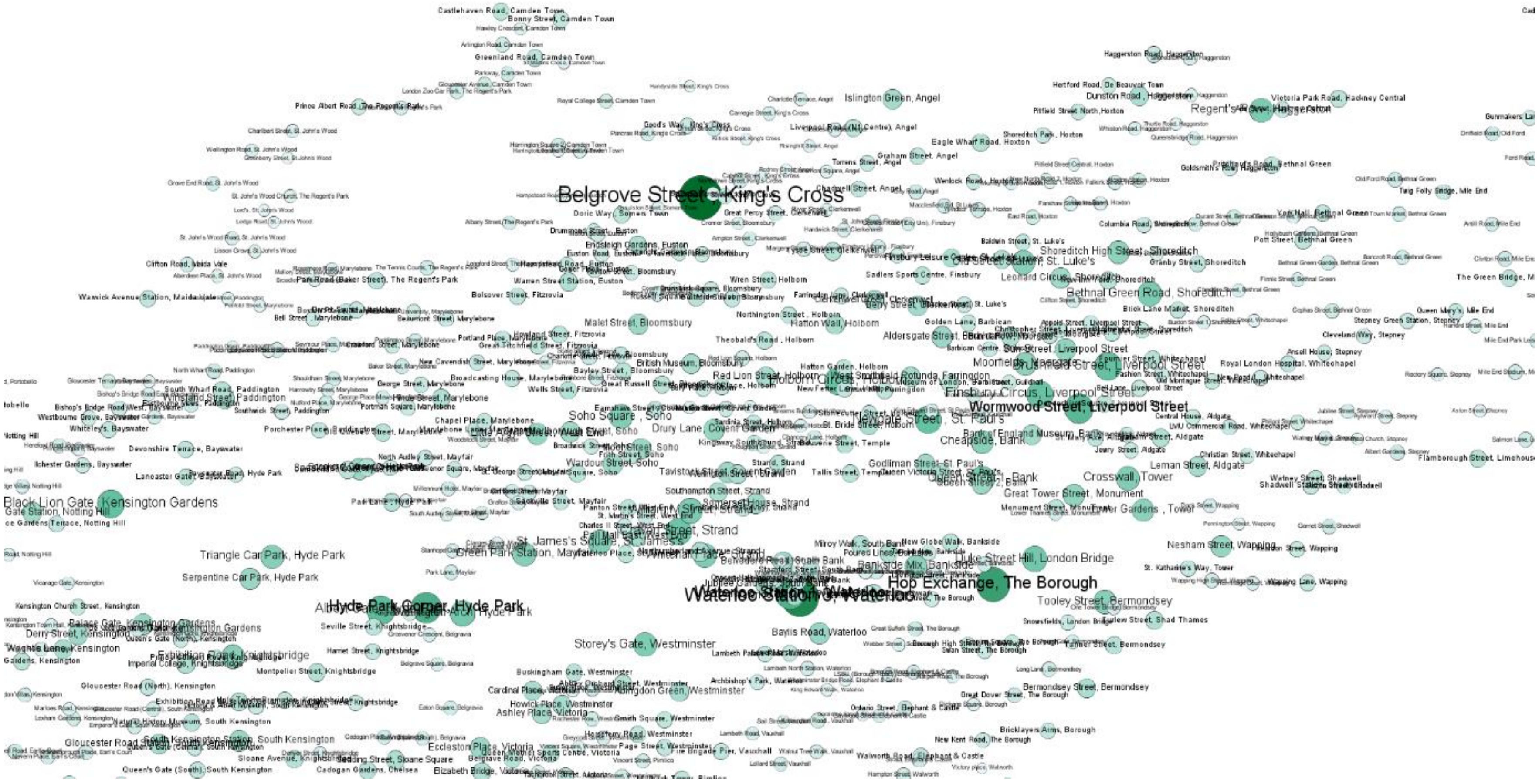
Gephi


Gephi 0.9.1 - geo.gephi
File Workspace Tools Window Help
Overview
Data Laboratory
Preview
Workspace 1
Workspace 2
Appearance
Nodes Edges
Unique Partition Ranking
Weight
Color:
Spline...
Apply
Layout
Geo Layout
Run
Scale 1000000.0
Latitude latitude
Longitude longitude
Projection Mercator
Center
Looping
Geo Layout
Presets... Reset
Preview Settings
Graph
Arial Bold, 32
Context
Nodes: 787
Edges: 368294
Directed Graph
Statistics
Filters
Reset
Library
Attributes
Equal
latitude Double (Node)
longitude Double (Node)
rank Float (Node)
Inter Edges
Intra Edges
Non-null
Partition
Partition Count
Range
Queries
Equal (rank)
Equal (rank) Settings
Value 0.15
Filter

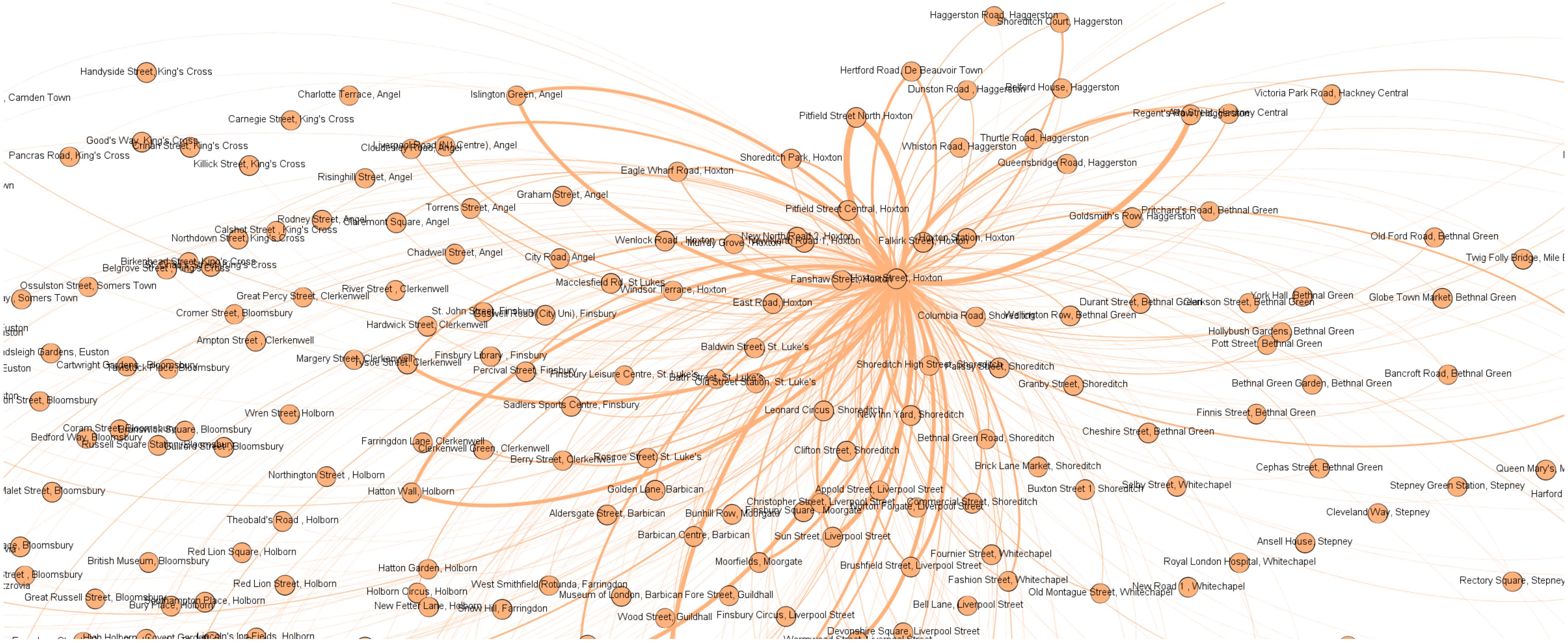
Results

- Belgrove Street, Kings Cross, ranked the highest. Also has the largest in-degree.
- Same finding from Pandas aggregation.
- But many locations have higher rank than would be predicted by in-degree alone.
- Saunders Road, Cubitt Town, has the highest increase in rank relative to other locations with higher in-degree.
- Hoxton Station has the highest decrease.

Results (overall)



Results (from Hoxton Station)



Interpretation?

- More consistent approach than simple aggregation; concept applied to the entire network, rather than a collection of ad-hoc stats.
- Also allows perspective from specific locations; would be hard to do without traversal approach.
- Initial findings seem similar overall, though – in terms of most influential / busiest locations.
- Does it buy us anything?

Interpretation?

- Analogy with internet browsing not exact
  - Cyclists do not randomly chose from a selection of links.
  - Cyclists only make one journey rather than a sequence of steps.
  - No “random reset”.
  - No equivalent of “likelihood of user (eventually) landing on a specific web page”.
  - => Cyclists != Web Surfers, in the analogy?

Interpretation?

- However, from the bike’s point of view
  - Markov model; each journey is a next step from wherever it currently finds itself.
  - Next step is “randomly” chosen, but will be more or less likely depending on previous journey counts from that location.
  - TFL do re-distribute bikes to other locations.
  - => Bikes == Web Surfers, journeys == links.
  - PageRank calculates the likelihood of a bike (eventually) ending up at a location?
- May be useful for network/capacity planning.