**Complexity of some operations**

Converting between COO, CSR and CSC: Linear

“All conversions among the CSR, CSC, and COO formats are efficient, linear-time operations.” <https://docs.scipy.org/doc/scipy/reference/sparse.html#usage-information\>

<https://github.com/scipy/scipy/blob/3b36a574dc657d1ca116f6e230be694f3de31afc/scipy/sparse/sparsetools/coo.h>

Transposing: Linear because it converts from CSR to CSC or creates a new COO behind the scenes, and those operations are linear.

Coo does coo\_matrix((self.data, (self.col, self.row)), shape=(N, M), copy=copy)

inverting self.col and self.row and the shape

<https://github.com/scipy/scipy/blob/v1.7.1/scipy/sparse/coo.py#L291-L299>

CSR and CSC call eachother’s constructors

<https://github.com/scipy/scipy/blob/v1.7.1/scipy/sparse/csr.py#L135-L145>

<https://github.com/scipy/scipy/blob/v1.7.1/scipy/sparse/csc.py#L108-L118>

Extracting a diagonal from a CSR -> O(nnz)

Src: <https://github.com/scipy/scipy/blob/3b36a574dc657d1ca116f6e230be694f3de31afc/scipy/sparse/sparsetools/coo.h#L31>

**Complexity**

Loading graphs: Depends how we do.

Time:

* In make\_symmetric O(E) since we iterate over edges
* Linear time to create a COO
* Linear time to transpose the matrix and add it to original

Space:

* Np.loadtxt() is O(E) since file is an edgelist
* Make\_symmetric O(E) when we convert to list of sets, then set, then list again
* O(E) again when building COO matrix

a) Degree distribution

laplacian() to get the degrees quickly looks Linear

<https://github.com/scipy/scipy/blob/47bb6febaa10658c72962b9615d5d5aa2513fa3a/scipy/sparse/csgraph/_laplacian.py#L86>

else sum on rows O(n) for CSR

Counter is O(n) to create since its basically a dict

Then its just a bunch of operations and single loops so O(n)

b) Clustering coefficients

Need to multiply A by itself 3 times.

CSR matrix multiplication: O(n\_row \* K^2 + max(n\_row, n\_col))

where K is the **maximum nnz** in a row of A and column of B.

This is max n-1 for a fully connected node, so O(n3), though in practice it might feel quicker since not all nodes have lots of connections. In fact, given the power law distribution of degrees, we know very few of them do.

Src: <https://github.com/scipy/scipy/blob/701ffcc8a6f04509d115aac5e5681c538b5265a2/scipy/sparse/sparsetools/csr.h#L542-L544>

c) Using Djistra’s algorithm with Fibonacci heaps

Time complexity is “approximately O[N(N\*k + N\*log(N))], where k is the average number of connected edges per node. The input csgraph will be converted to a csr representation.”

Since our average degrees are around 3-4 (?) this is O(n^2 \* log N )

<https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csgraph.shortest_path.html>

d) Using csgraph.connected\_components()

The documentation refers [1] which gives this algorithm Θ(v + e) where v = # nodes, e = # edges

[1] D. J. Pearce, “An Improved Algorithm for Finding the Strongly Connected Components of a Directed Graph”, Technical Report, 2005

e) Eigenvalues

We used the scipy.sparse.linalg.eigsh() function, which according to [1] uses the Implicitly Restarted Lanczos Method to find the eigenvalues.

According to Wikipedia (better source?) O(dn^2) where d is the average number of nonzero elements in a row.

So for the Laplacian, which has all non-zero values, O(n3)

<https://github.com/scipy/scipy/blob/v1.7.1/scipy/sparse/linalg/eigen/arpack/arpack.py#L1351-L1692>

<https://en.wikipedia.org/wiki/Lanczos_algorithm>

connected compo uses

<https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.102.1707&rep=rep1&type=pdf>

<https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csgraph.connected_components.html#scipy.sparse.csgraph.connected_components>