Implementation of Sublinear Clustering Algorithm (Rough Draft)

By James Camacho and Joseph Camacho

Overview

We implement the algorithm "Sublinear Time and Space Algorithms for Correlation Clustering via Sparse-Dense Decompositions" by Sepehr Assadi and Chen Wang. This clusters data in a (+/-)-labeled graph, where pluses mean two vertices are correlated and minuses mean anticorrelation. For example, a graph of all English words could have pluses between synonyms and minuses between antonyms. In other datasets, such as document clustering, these graphs can be incredibly large and dense, leading to a superlinear number of edges compared to the vertices. Even a single pass through the edges can be infeasible. This algorithm presents an approximation scheme that is $\tilde{O}(n)$ in the number of vertices, often sublinear in the size of the dataset.

It requires access to the plus-labeled edges in an adjacency list, and works as follows:

1. Sample a constant number of edges from each vertex.
2. Choose a random (sublinear) number of vertices and query every edge of these vertices.
3. Filter these vertices using the sampled edges from step one to find the densest ones.
4. Create candidates for the clusters based on the dense vertices and their neighbors.
5. Find approximate clusters from the candidates.

If the constants are chosen right, there should be a high probability that the candidate clusters in step four form a laminar group, i.e. each candidate is either disjoint or a subset to any other candidate. Then, step five consists of assigning each vertex to the largest candidate it's contained in. If it isn't contained in any candidate clusters, it is assumed to be pretty isolated and assigned to its own cluster.

We can define a cost for the clustering---each time two vertices are put in different clusters but have a plus-edge between them, the cost is increased by one. Similarly if two vertices share a minus-edge but get put in the same cluster, the cost increases by one. It turns out the minus-edges are not too relevant, as two vertices with a minus-edge should only be clustered together if they share a lot of neighbors in the plus-subgraph, in which case the cost would increase whether or not they get clustered together. So, by only using the plus-subgraph, it is possible to determine good clusters with an $O(1)$ approximation to the optimum.

The accuracy of the algorithm depends on the details. How many edges do you sample from each vertex? What is the probability you sample all the edges for a particular vertex? What bounds do you use to filter dense vertices?

The paper suggests sampling $t = O(\varepsilon^{-2} \log n)$ vertices for some constant ε. This ε is never specified in the paper, but certain theoretical results of theirs require that $\varepsilon < 1/360$.

In step two, a vertex v is chosen with probability $O(\log n / \deg(v))$. Note that this rewards sparser vertices—in a complete graph with n vertices you would only expect $O(\log n)$ vertices, and thus $O(n \log n)$ edges to be sampled, while a tree would have nearly every vertex chosen. Overall, steps one and two only require us to sample $\tilde{O}(n\varepsilon^{-2})$ edges.

The filtering in step three is a little more complicated. First, all vertices that have much denser neighbors should be filtered out. This is according to the formula

$$\varepsilon \deg(v) < |\{u \text{ neighbors } v : \deg(u) > (1+\varepsilon)\deg(v)\}|.$$

Then, sparse vertices are filtered out. Call the *low* neighbors of v the vertices u where

$$\deg(u) < (1+7\varepsilon)\deg(v).$$

A low neighbor u is *isolated* if

$$|\text{sampled neighbors}(u) \cap \text{low}(v)| < (1-4\varepsilon)t,$$

i.e. very few of its neighbors are also low neighbors of v. If a large fraction, at least $2\varepsilon \deg(v)$, of the low neighbors of v are isolated, then v is considered sparse. We're left with only the densest vertices from our sample in step two.

From here, we use some more magic numbers to build candidate clusters. For each dense vertex v, we build a candidate set, including a low neighbor u if

$$1 + 22\varepsilon > \frac{\deg(u)}{\deg(v)} > \frac{(1-67\varepsilon)t}{|\text{sampled neighbors}(u) \cap \text{low}(v)|}.$$

Recall that there are t sampled neighbors of u, so the right inequality means that most neighbors of u are also low neighbors of v. In most of our analysis we have $\varepsilon > 1/67$, so this inequality doesn't do much, but it will serve to increase accuracy as ε gets very small.

In this paper we explore more how the choice of ε and constant multiplier in $t = O(\varepsilon^{-2} \log n)$ affect the operation count and accuracy of the algorithm. We answer questions like, exactly how close are the candidate clusters to a laminar set family? How many intersections can we expect?

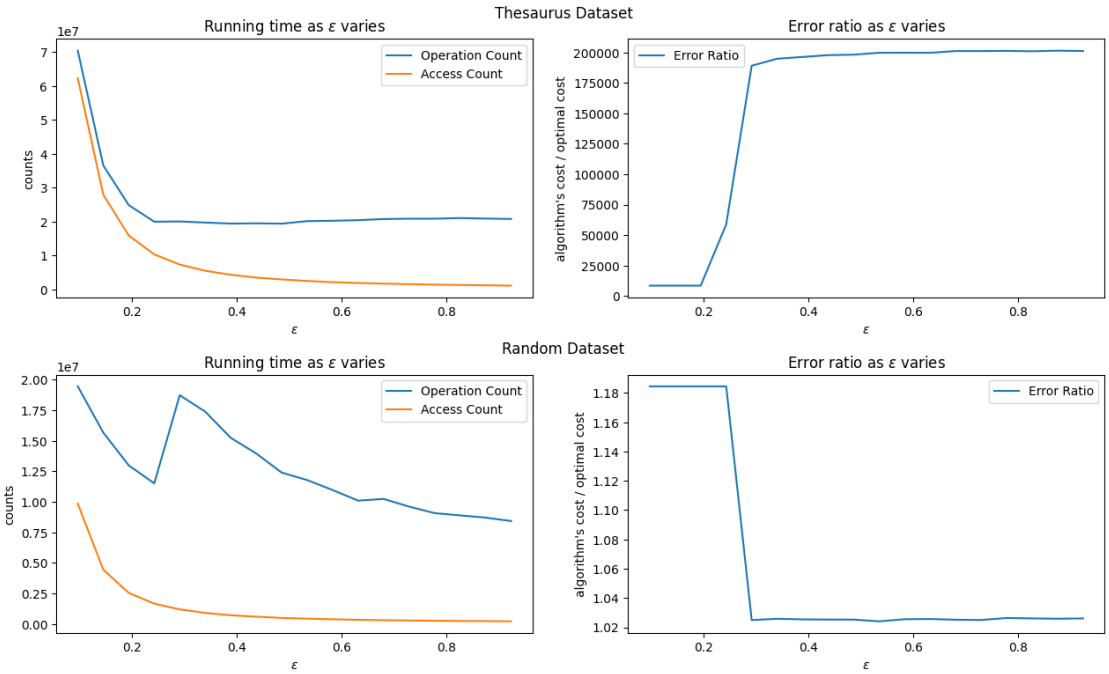
Do we *really* need $\varepsilon < 1/360$ in practice, or would larger values like $\varepsilon = 1/64$, $\varepsilon = 1/2$ or $\varepsilon \approx 1$ fare equally well? How do sparser or denser datasets change these results?

Choice of ε

Assadi and Wang's main result is that their algorithm results in a constant-approximation of the optimal clustering in sublinear time. However, the quality of this constant depends very much on their choice of ε. In particular, they showed that the constant is roughly proportional to ε^{-2}. This makes restricting ε to less than $1/360$ (as certain theoretical results of theirs require) suboptimal: If the optimal clustering of a graph has a single incorrect correllation, using their clustering algorithm with $\varepsilon = 1/361$ could have on the order of over 100,000 incorrect edges. This is a terrible approximation!

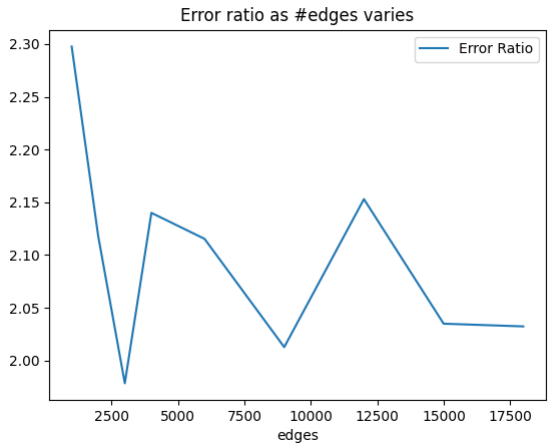
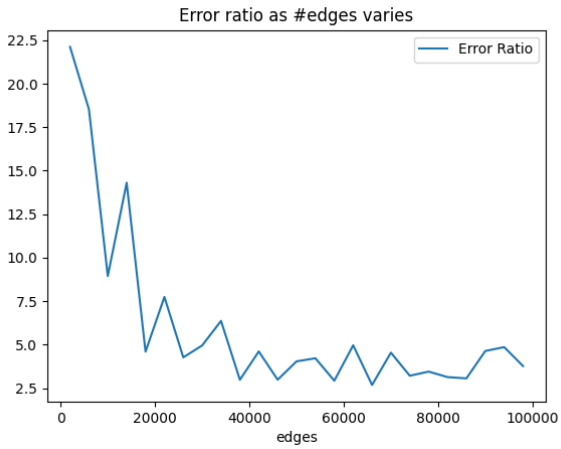
Luckily, in practice ε can be raised to much larger values while maintaining sublinear time in edges, vastly improving the approximation of the clustering. In order to test this, we use two datasets. The first is a graph of English words with plus-edges between synonyms and minus-edges between antonyms. The second is a random graph with ten thousand vertices, one million edges, and ten clusters; edges within a cluster are assigned a plus while edges between clusters are labelled with a minus, however we randomly flip between 10% and 45% of these correlations.

For both datasets running time improved as ε increased, as fewer edges needed to be considered. However, the thesaurus dataset differed from the random dataset with the errors. As ε transitioned from less than 0.25 to more than 0.25 we saw a pronounced increase in the number of errors on the real-world data, while a marked improvement in accurracy for the random dataset. Note that determining the optimum for the thesaurus dataset is NP-hard, so we instead give the error count.



This difference likely arises as there are only ~8,000 antonyms compared to ~200,000 synonyms in the thesaurus. Putting all words into one cluster is guaranteed to have very few errors, while separating into multiple clusters risks synonyms being in different clusters.

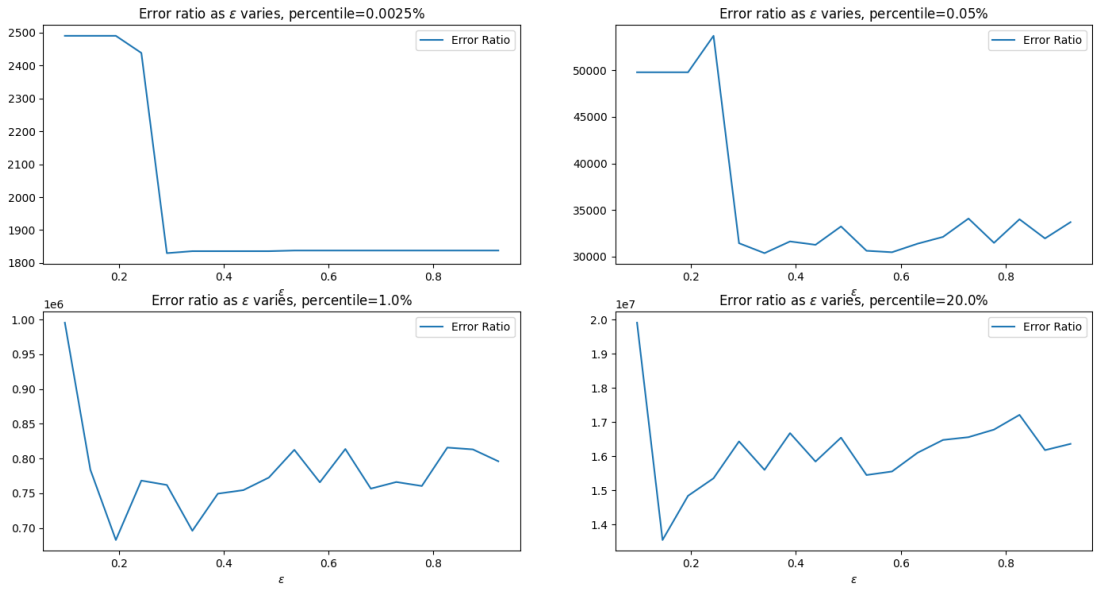
For the random dataset, we see this rapid improvement at $\varepsilon = 0.25$ even as we change the fraction of edge correlations that are flipped, how many clusters there are, or the size of the graph. For example, the below graphs show the algorithm run with $\varepsilon < 0.25$ and $\varepsilon > 0.25$ and a varying number of edges. Even with very few edges we have over four times the error when $\varepsilon < 0.25$, and as it gets denser this increases to over twenty times!

$\varepsilon < 0.25$	$\varepsilon > 0.25$
	

To get a denser model based on real-world data, we create a new dataset using word vectors. It is well known their dot products are a measure of correlation, with very positive dot products being highly correlated while negative dot products are anticorrelated, so we assign plus-edges

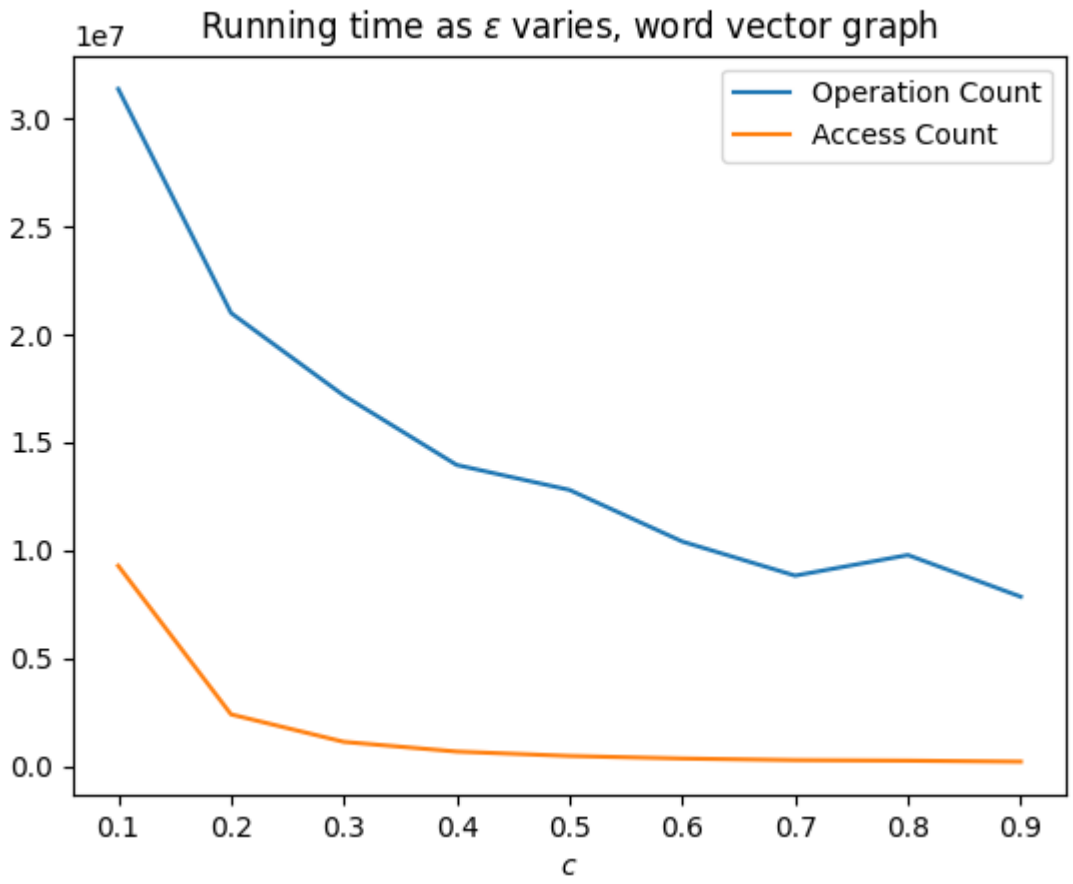
to the top percentiles of these dot products, and minus-edges to the bottom. Changing this fraction allows for sparser or denser graphs.

For all levels of sparsity, we see an error trend similar to that of the random graphs:



For both the random graph and the graph based on word vectors, the best clustering occurs when ε is slightly larger than 0.25.

Likewise, the running time improves as ε increases:



This suggests several things, most notably that filtering by isolated neighbors is actually detrimental to the performance of the algorithm! Although it's theoretically necessary to do this filtering to gurantee an $O(1)$ approximation (as the thesaurus dataset somewhat shows), in practice it's not only faster to not do so, but also more accurate.

In conclusion, real world data often looks similar to the random dataset we built, and for these types of data, setting ε to be slightly more than 0.25 is a good choice.

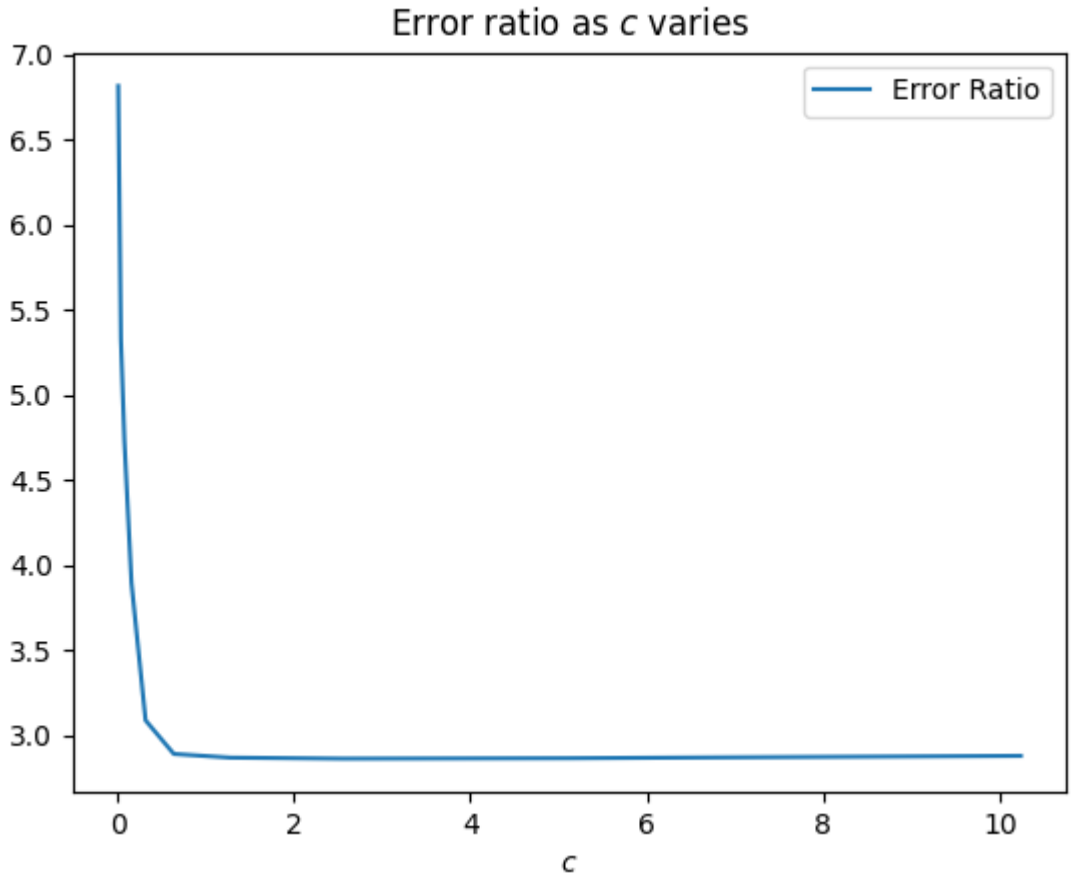
Choice of c

In their paper, Assadi and Wang state that "there is an absolute constant $c > 0$" for which their algorithm works, but they never specify exactly what c has to be. In fact, they only imply what restrictions c might have by constraints on other variables. These constraints imply that c to be on the order of at least ε^2. Since ε^2 is rather small, this means that c can be just about anything.

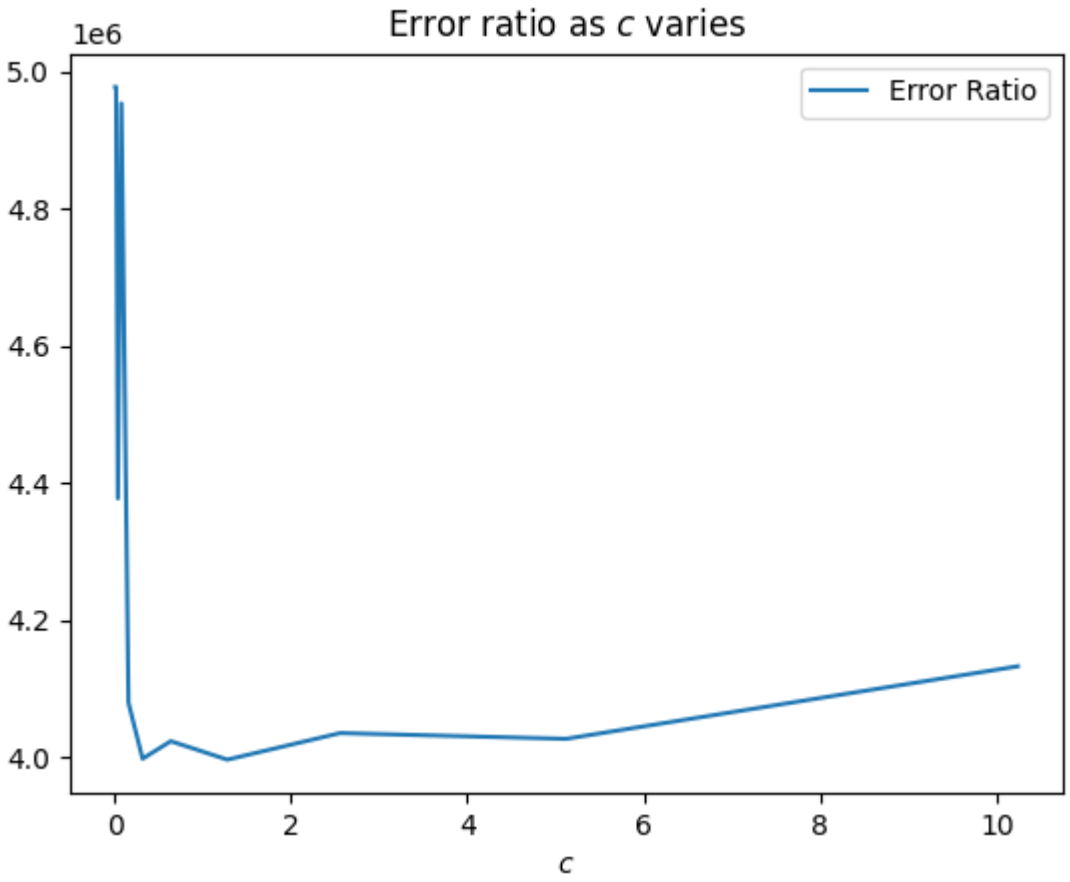
Nevertheless, setting $c = \varepsilon^2$ is hardly useful. While lowering c speeds up the algorithm by requiring fewer graph accesses, it also lowers its performance because the algorithm has less information to make use of. In this section, we explore the tradeoffs of what c should be, and demonstrate that $c \approx 2$ yields good results with competitive results.

In the previous section, we showed that a good choice of ε is not much more than 0.25. For this section, we set $\varepsilon = 0.3$.

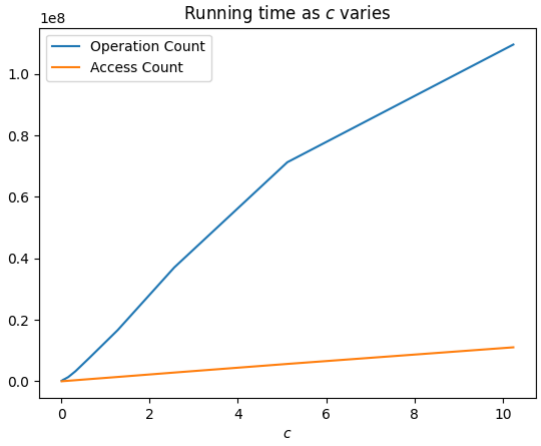
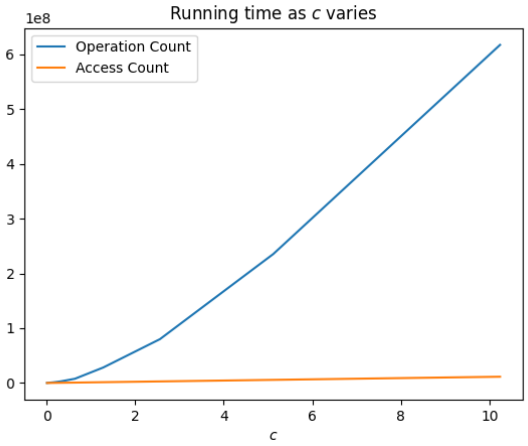
Using the random graph, we get the following approximations as c varies:



Using the word vector graph gives a similar chart:



In both cases, the running time scales linearly with c:

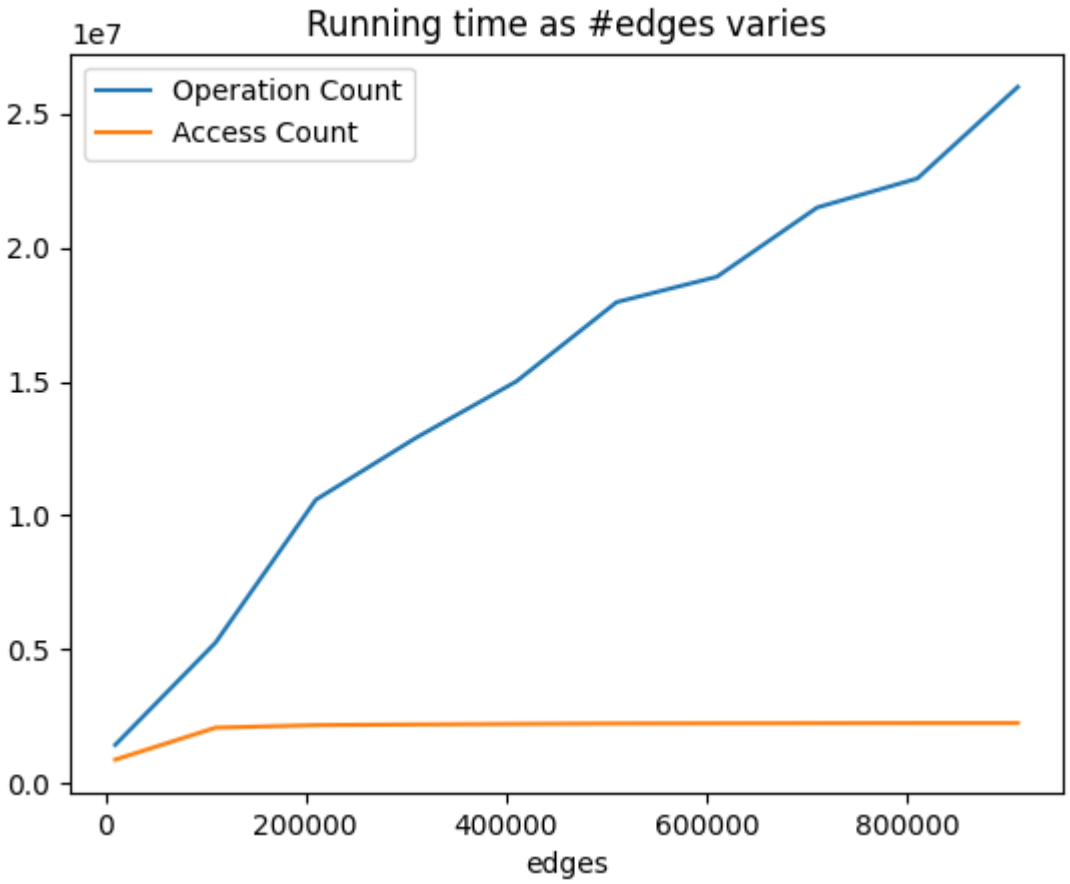
Random	Word Vector
	

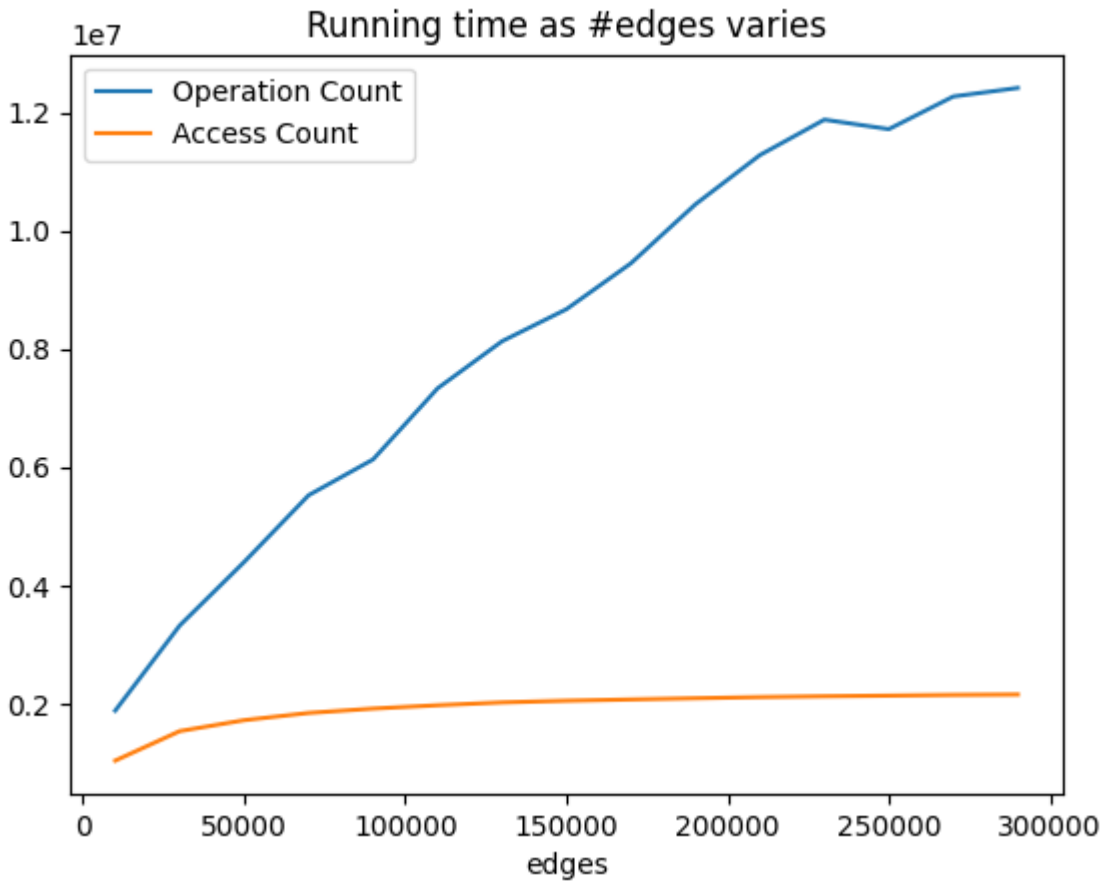
A good trade off, then, between runtime and error ratio appears to be setting $c \approx 2$.

Sublinear Time

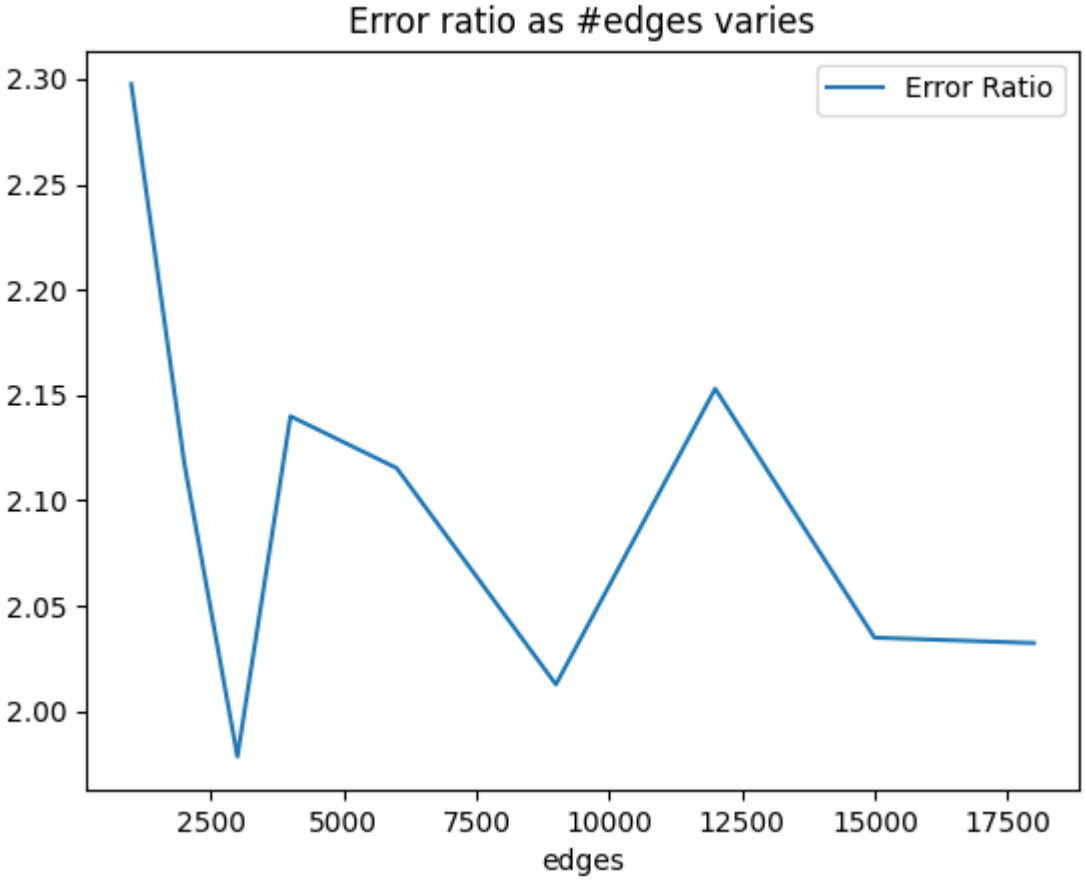
It's easy enough to theoretically show that the algorithm described above is sublinear in the number of edges no matter what the specific choices of ε and c are. (Indeed, Assadi and Wang's paper is important not because their algorithm is sublinear in time, but because it has $O(1)$ competitiveness.) In this section, we demonstrate this by analyzing the runtime for fixed $\varepsilon = 0.3$ and $c = 2$.

TODO: Uh oh James! This doesn't actually work! The graphs appear to have linear time!!!
Random	Word Vector


Running time as #edges varies
1e7
Operation Count
Access Count
2.5
2.0
1.5
1.0
0.5
0.0
0
200000
400000
600000
800000
edges


Running time as #edges varies
1e7
Operation Count
Access Count
1.2
1.0
0.8
0.6
0.4
0.2
0
50000
100000
150000
200000
250000
300000
edges

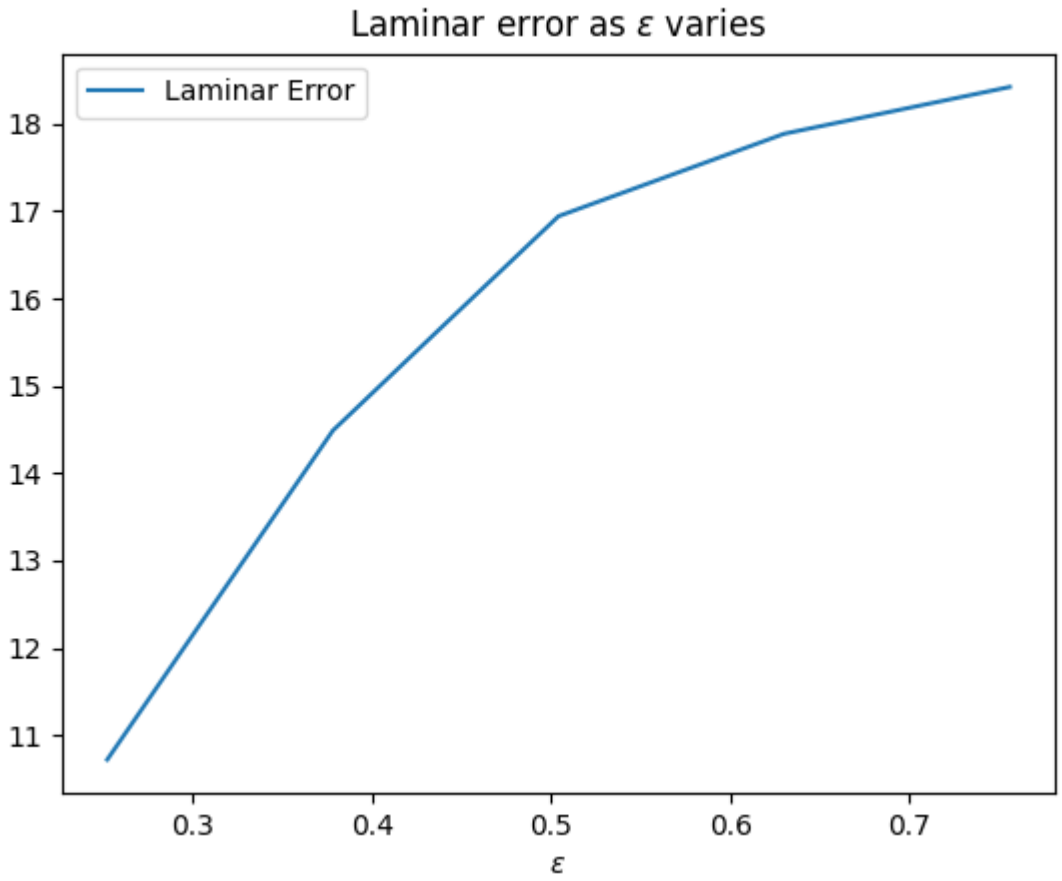
I think this is where we need the filter part! We have sublinear time for low ε. The following graph is for where $\varepsilon = 0.01$:

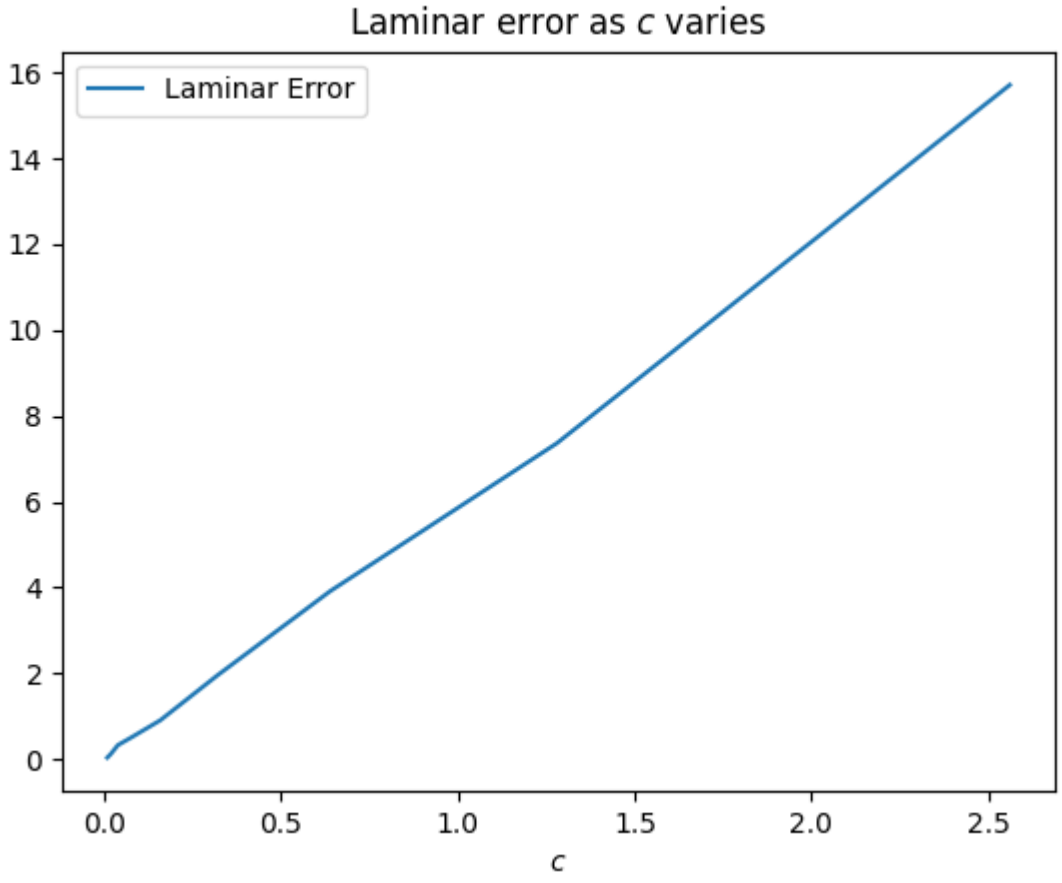


Laminar Candidate Clusters

Assadi and Wang make the claim that the candidate clusters have a high probability of forming a laminar set family. Every pair of candidates should either be disjoint, or one is a subset of the other. Suppose two candidates do not satisfy this laminar property. Their symmetric difference is all elements contained in exactly one subset but not the other, and so the sum of the symmetric differences is a measure of how far off from laminar the candidates are. Normalizing by number of vertices and number of candidate sets gives the following graphs:

Random graph, $V = 10^4, E = 10^6, \text{clusters} = 10, \text{flip} = 10\%$.
$c = 2$ is fixed, ε varies.	$\varepsilon = 0.3$ is fixed, c varies.

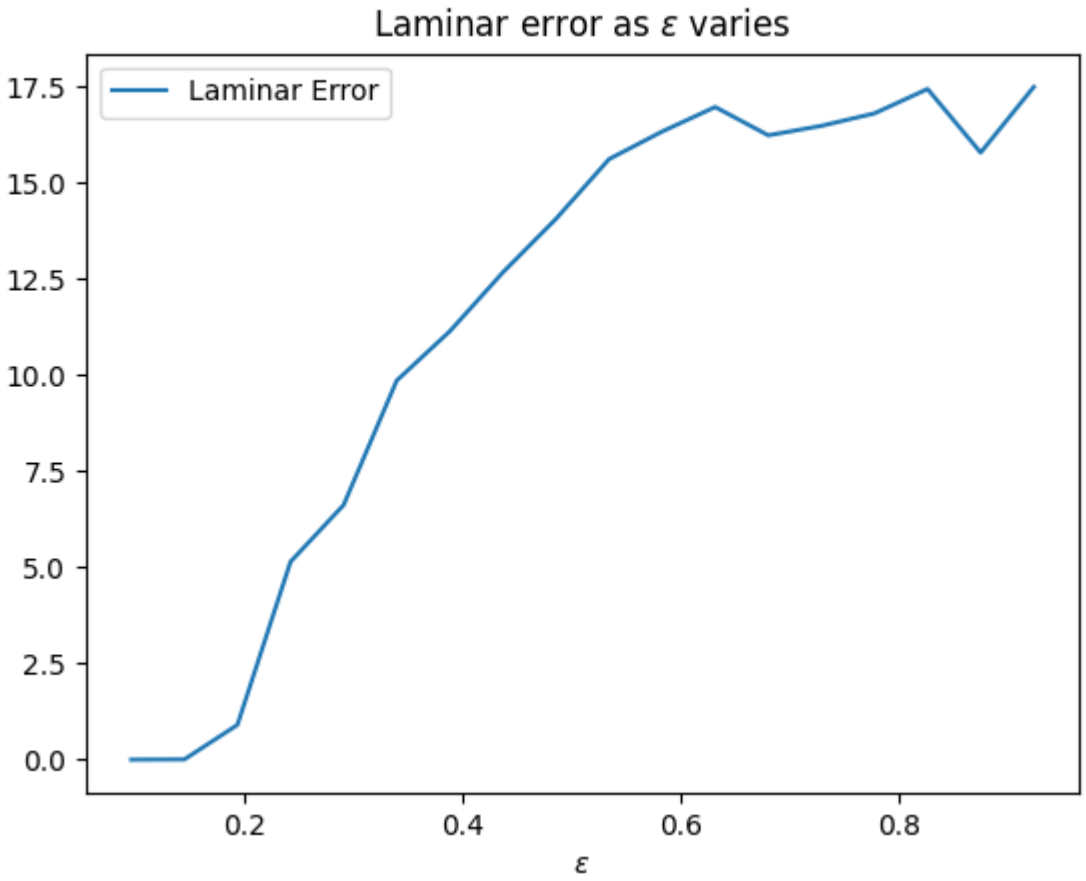

Laminar error as ε varies
Laminar Error
18
17
16
15
14
13
12
11
0.3
0.4
0.5
0.6
0.7
ε

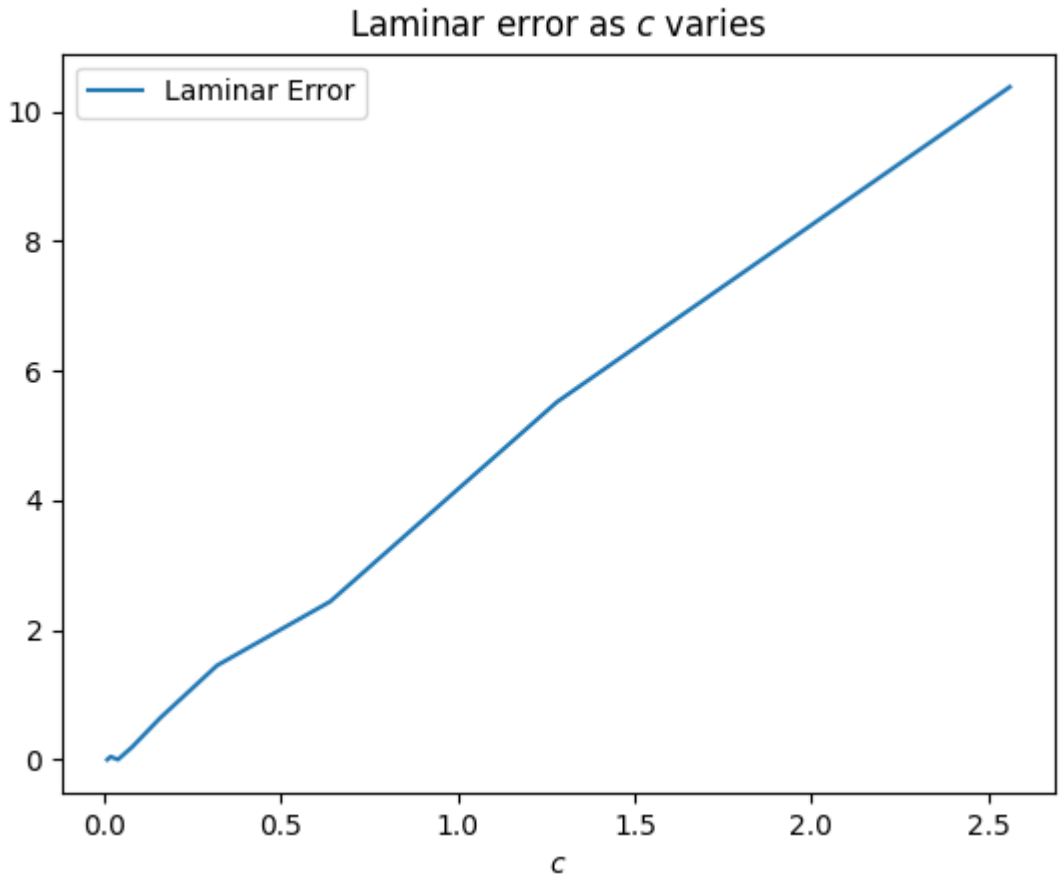

Laminar error as c varies
Laminar Error
16
14
12
10
8
6
4
2
0
0.0
0.5
1.0
1.5
2.0
2.5
c

Word vector graph, $V = 10^4, E = 5\%$.

$c = 2$ is fixed, ε varies. | $\varepsilon = 0.3$ is fixed, c varies.

:----:|:----:





When ε is very small there are no candidates, but otherwise we see a linear increase in laminar error as c increases, and approximately sublinear for ε. From our earlier analysis, for graphs of this size we want $\varepsilon \approx 0.3$ and $c \approx 2$, which would give the average vertex at most ten potential clusters, around 1% of the total number of clusters. This is pretty good. It would be interesting for a future paper to explore a recursive strategy using these candidate clusters, instead of just assigning vertices through a loop of the largest candidates.

Conclusion

In this paper, we implemented Sepehr Assadi and Chen Wang's algorithm for correlation clustering and empirically showed that their theoretical results held true.

References

"Sublinear Time and Space Algorithms for Correlation Clustering via Sparse-Dense Decompositions"