APPLICATIONS OF PERSISTENT HOMOLOGY TO IDENTIFYING CLUSTERS IN GRAPHS

M. RIZOIU, M. SKILLETER, K. TURNER

ABSTRACT. We study practical applications of persistent homology to identifying bridges in graphs. These graphs arise when analysing social networks, where bridges represent users who are members of multiple communities. In this paper, we outline several variations on an algorithm that utilises persistent homology and a metric on persistence diagrams, called the Wasserstein Distance, to identify cluster points. We also provide a mathematical justification for the efficacy of this algorithm and discuss ways in which it can be optimised to be applied to large data sets.

1. INTRODUCTION

Society is just beginning to see the effect that a group of dedicated diffusers of malicious information can have. Private and state agencies are using platforms such as Twitter to push their ideas, and the problem of identifying these information distributors is becoming more important. In this paper, we will demonstrate how techniques from the area of topological data analysis can be used to address this problem. Specifically, we will use ideas from persistent homology to identify bridges in graphs.

The field of topological data analysis (TDA) has been rising in prominence since the start of the 21st century. It utilises ideas from an area of mathematics called algebraic topology to draw inferences about large data sets by studying their structure when considered as a geometric object. As TDA becomes more widespread, various programming languages have added support. A full implementation of our algorithm in the language Julia can be found at **https://github.com/OuroborosOfLife/Zigzag-Persistent-Homology/blob/master/Code/classifyingBridges.jl**, making use of the package Eirene.

We will now give the important definitions, as well as state some important theorems without proof.

1.1. **Graph Theory.** A graph is a tuple (V, E, μ) with V a set of vertices, E a set of edges and $\mu : V \cup E \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ a weight function. An edge between vertices v_0 and v_1 will be denoted by (v_0, v_1). A graph is called *undirected* if $(v_0, v_1) = (v_1, v_0)$ for every edge. All the graphs considered in this paper will be undirected and will have finite vertex and edge sets.

An important example of a weight function is the *shortest path* weight function, which can be computed using Djikstra's Algorithm. Fix a vertex v and for any other vertex w, define $\mu_v(w)$ to be the length of the shortest path from w to v. For an edge (v_0, v_1), we define $\mu_v((v_0, v_1)) = \max\{\mu_v(v_0), \mu_v(v_1)\}$.

The *path-component* of a vertex v is the subgraph of all vertices that can be reached from v. A graph is called *connected* if it only has one path-component. Notice that if v and w are in different path components then the distance between them under the shortest path weight function is ∞.

1.2. **Homology.** We will only give definitions for homology in the case of a graph, but recognise that these definitions can be generalised to any space. For a completely general definition of homology, see [1].

For a graph $X = (V, E, \mu)$, we define two groups $C_0(X)$ and $C_1(X)$ by

$$C_0(X) = \mathbb{Z}\{V\}$$
$$C_1(X) = \mathbb{Z}\{E\}$$

Here the notation $\mathbb{Z}\{V\}$ means that we take finite formal sums of elements in V with coefficients in $\mathbb{Z}$. For example, if $V = \{v_0, v_1, v_2\}$ then $5v_0 - v_1 + 2v_2$ is an element of $C_0(X)$. The notation $\mathbb{Z}\{E\}$ is entirely analagous. Elements of $C_0(X)$ are called 0-chains and elements of $C_1(X)$ are 1-chains.

We next define the boundary operator $d_1 : C_1(X) \to C_0(X)$. For an edge $e = (v_0, v_1)$, we set $d_1(e) = v_1 - v_0$ (considered as a formal sum in $C_0(X)$). For an arbitrary 1-chain, we can now define

$$d_1 \left(\sum_i c_i e_i \right) = c_i \sum_i d_1(e_i)$$

Observe that we have defined d_1 on the basis E and extended it linearly to arbitrary 1-chains. This means that d_1 is a group homomorphism and so we can consider the subgroups $\ker(d_1)$ and $\operatorname{im}(d_1)$. The homology groups $H_0(X)$ and $H_1(X)$ are then given by

$$H_0(X) = C_0(X)/\operatorname{im}(d_1)$$
$$H_1(X) = \ker(d_1)$$

Intuitively, one can think of $H_n(X)$ as measuring the number of n-dimensional "holes" in X. This idea is illustrated by the following example.

Example 1.1. The circle S^1 can be thought of as a graph with a single vertex v and a single edge $e = (v, v)$ which is a loop from v to itself. To compute d_1, we evaluate it on the basis element e to see that $d_1(e) = v - v = 0$. Thus d_1 is the zero map, so $\ker(d_1) = C_1(S^1) = \mathbb{Z}$ and $\operatorname{im}(d_1) = 0$. Therefore $H_0(S^1) = H_1(S^1) = \mathbb{Z}$. The single power of $\mathbb{Z}$ in $H_1(S^1)$ indicates that there is only one 1-dimensional "hole" in the circle which is exactly what one would expect. As we will see in the next theorem, the single $\mathbb{Z}$ in $H_0(X)$ shows that S^1 is connected.

A 0-dimensional hole is a gap between vertices that cannot be traversed, so one would hope for a relationship between the number of path-components and $H_0(X)$. Such a relationship is given by the following theorem.

Theorem 1.2. *Suppose X is a graph with m distinct path-components. Then $H_0(X) = \mathbb{Z}^m$.*

This gives us an easy method for computing $H_0(X)$ by simply counting path-components. There is one other relation, however, that will significantly simplify our calculations of homology.

The Euler characteristic χ is a well-known graph invariant defined as $\chi = |V| - |E|$. Using homology, it can be shown that:

Theorem 1.3. *The Euler characteristic of a graph X is also given by $\chi = \operatorname{rank}(H_0(X)) - \operatorname{rank}(H_1(X))$. Here rank is the function defined by $\operatorname{rank}(\mathbb{Z}^m) = m$.*

We already know how to compute $\text{rank}(H_0(X))$: by Theorem 1.2, this is the number of path-components in X. Because $H_1(X)$ is the kernel of a group homomorphism between free abelian groups, it is always a free group (meaning there is no torsion) so its rank fully determines $H_1(X)$. Equating the two above expressions for χ and rearranging shows that $\text{rank}(H_1(X)) = m - |V| + |E|$ where m is the number of path-components in X. Thus the above two theorems give us a constant time method for computing homology.

1.3. **Persistent Homology.** Building on homology is persistent homology. This is the idea that more information can be obtained from a graph by studying how $H_0(X)$ and $H_1(X)$ vary over time. To formalise this, we make use of filtrations. A filtration of a graph X is a finite nested sequence of subgraphs $X_0 \subseteq X_1 \subseteq X_2 \subseteq ... \subseteq X$. Essentially, a filtration is an inductive way to build the graph one stage at a time.

Using the shortest distance function defined earlier, we have a canonical way to obtain a filtration of X associated to each vertex v. We set X_0 to be the point v, X_1 to be the subgraph of vertices distance 1 away from v and so forth. In general, X_d is the subgraph of vertices distance d or less from v. Figure 1 below shows an example filtration of this sort, with v the blue vertex and the colour gradually increasing in intensity as we radiate outwards from v.
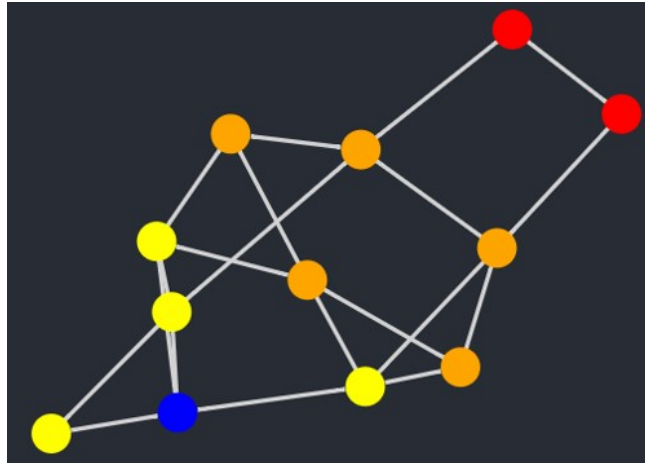

FIGURE 1. Example Filtration Using Shortest Path Function

Once we have a filtration, we can build what are called the persistence barcodes. There is a persistence barcode associated with each dimension n; the n-dimensional persistence barcode tracks the births and deaths of n-cycles (the elements of $H_n(X)$) as the filtration progresses. For example, because $H_0(X)$ measures path-components, the 0-dimensional barcode tracks the births and deaths of path-components. This means that if two disjoint path-components become connected by a bridge at some stage in a filtration, one of the 0-cycles will die at this stage. Because $H_1(X)$ measures 1-dimensional holes, one can think of the 1-dimensional persistence diagram as tracking the births and deaths of loops in X.

One important feature which makes persistence diagrams so useful is that there is an associated metric, called the Wasserstein distance (in general this is only a pseudo-metric, but it will be a true metric for diagrams considered in this paper).

The Wasserstein distance between two persistence diagrams A and B is given by

$$W_{p,q}(A, B) = \inf_{\substack{\eta:A\to B \\ \text{a bijection}}} \left(\sum_{a\in A} \|\|a - \eta(a)\|\|_q^p \right)^{1/p}$$

The values p and q are arbitrary positive integers (also possibly ∞) but for computational efficiency we will always take $p = q = 2$.

1.4. **Bridges in Graphs.** The purpose of this paper is to develop an algorithm that identifies bridges in graphs, so we must make clear what is meant by a bridge. This is difficult to do rigorously, but for intuition one should think of the following: a bridge between two clusters is a vertex which is connected to at least one vertex from each of the clusters.

It is important to emphasise that in our definition, a bridge is a vertex and not an edge as terminology sometimes varies across the field. We will not strictly adhere to the above definition of a bridge, but it is a useful starting point. As the reader sees example graphs, they will develop an understanding of what is meant by a bridge in a graph.

Conversely, a vertex which is not a bridge is called a cluster point and a collection of such cluster points is called a cluster. It is immediate that finding all bridges is equivalent to finding all cluster points and this will prove to be an easier problem.

Figure 2 depicts six example graphs with bridges and cluster points distinguished by colour. Throughout this paper, unless specified otherwise, we will take the convention that red vertices are bridges and blue vertices are cluster points.
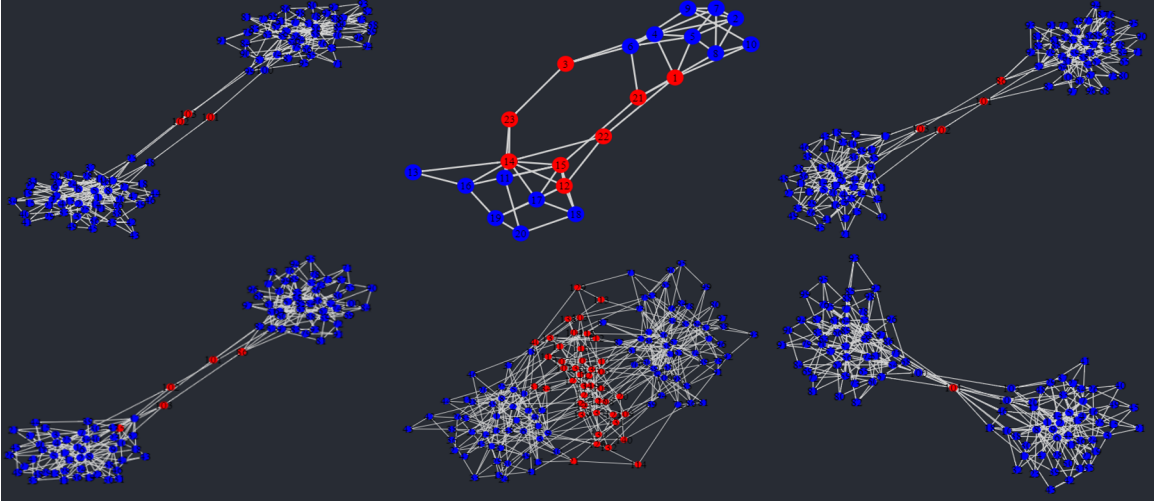

FIGURE 2. Graphs with Bridges Distinguished

2. The Algorithm

Here we will describe the development of the algorithm; for a summary, see the Discussion. Before we do, however, we make the following observation. A vertex v being a bridge is a local property in the sense that it only depends on the path-component of v. This means that we may assume without loss of generality that the graph X is connected by simply running the algorithm on each path-component of X separately. This is the first instance of parallelizability in the algorithm, since the existence of bridges in one path-component does not in any way affect bridges in other path-components.

2.1. **Early Attempts.** Continuing with this idea of locality, we describe our initial test for bridges which we shall call the "neighbourhood test". The key point is that a vertex v should be a bridge only if it is a bridge once we restrict to some small neighbourhood. This is because vertices far from v should not affect the behaviour of v. Having restricted to a suitably small neighbourhood, we can now use more classical tests for bridges such as checking if removing v disconnects the graph. As a first choice of small neighbourhood, consider the subgraph of vertices all directly adjacent to v. Figure 3 shows the neighbourhood test (with this choice of neighbourhood) applied to a typical graph.
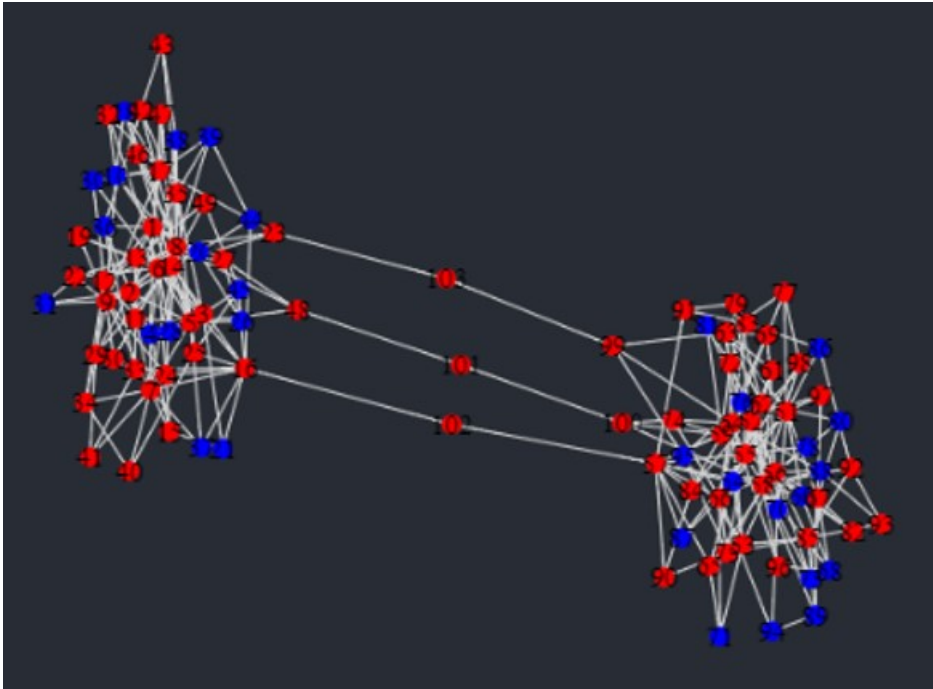

FIGURE 3. Neighbourhood Test Applied to Example Graph

While it is better to have false positives than to eliminate bridges, the neighbourhood test incorrectly identified a large number of cluster points as bridges. This problem can be resolved by taking a larger neighbourhood around each vertex, since removing v from a larger neighbourhood is less likely to disconnect the graph. However, this method often eliminates parallel bridges such as those shown in Figure 3 because the parallel bridges are included in larger neighbourhoods, so removing v no longer disconnects the graph.

One possible solution to this is to make the radius of the neighbourhood around each vertex dependent on v. This means that smaller radii can be chosen for parallel bridges to avoid the problem. However, there is no obvious way to choose a meaningful radius (doing so is almost identifying whether a vertex is a bridge or a cluster point). Instead, we introduce a generalisation of the neighbourhood test called the annulus test.

Define the closed annulus $\Delta(v, a, b) = \{w \in V \mid a \leq \mu_v(w) \leq b\}$. Rather than choosing a radius dependent on v for use in the neighbourhood test, we will check if $\Delta(v, a, b)$ is disconnected for sufficiently many a and b. Since the neighbourhood test applied to the neighbourhood of radius r about v is simply checking if $\Delta(v, 1, r)$ is disconnected, the annulus test is a true generalisation. Figure 4 shows the results of the annulus test applied to an example graph.
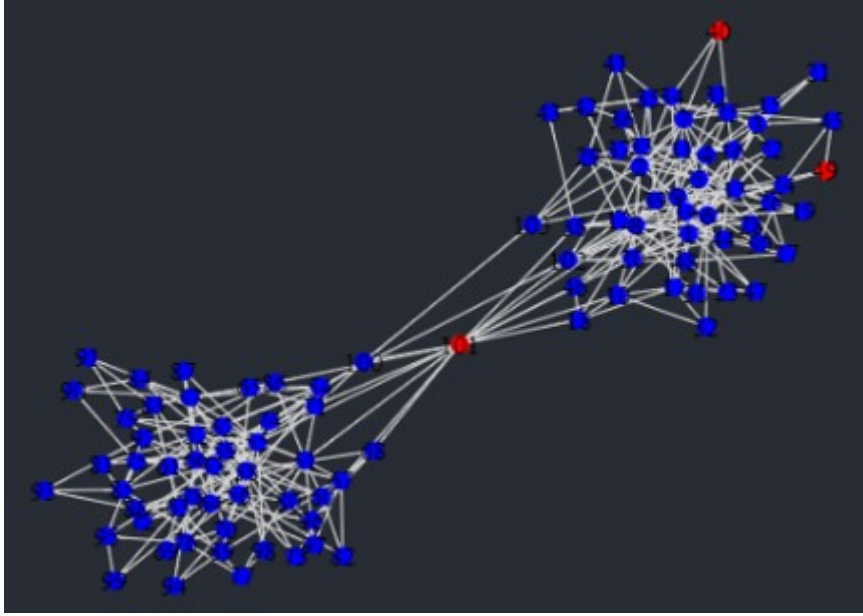

FIGURE 4. Annulus Test Applied to Example Graph

While the annulus test was noticeably more accurate than the neighbourhood test, it still incorrectly identified some cluster points as bridges (such as those in the upper right of Figure 4). This was surprising because, in general, the vertices which were classified incorrectly clearly belonged to clusters. To rectify this problem, the key observation was that the annulus test was not marking bridges as cluster points (it was only giving false positives). This meant that we could simply add further tests, now with the additional information of which nearby vertices were cluster points.

By considering graphs such as Figure 4, one might hope to find a meaningful way to say that two nearby vertices "look similar" so that if one has been identified as a cluster point then the other should be too. Persistent homology provides such a method by considering the persistence barcodes generated by the shortest path filtration. In a cluster, there are a large number of 1-cycles being born in quick succession because there is a high density of edges forming loops. For a bridge, though, there is an initial lull before a large number of births. Figure 5 shows column plots for the number of 1-cycles born each time the filtration progresses one step further from v. The left diagram corresponds to a typical cluster point and the right to a bridge.
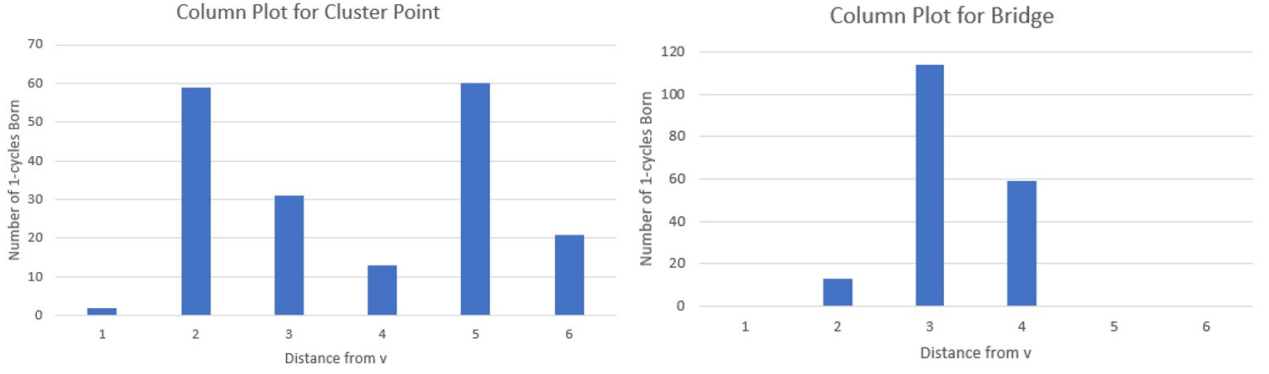
FIGURE 5. Column Plots for Number of 1-cycles Born vs. Distance from v

The Wasserstein distance allows us to compare the 1-dimensional persistence diagrams to find cluster points that may have been misidentified in our initial search for bridges. This is a significant simplification because it reduces our search to an extremal problem: we need only identify the vertices which are the most densely clustered and then use the Wasserstein distance to identify those neighbours which are also cluster points.

There is one problem with this approach, which is that highly symmetric graphs can generate similar persistence barcodes for vertices which are quite far apart. There are cases where the persistence diagrams corresponding to vertices can be very close under the Wasserstein distance even though the vertices themselves are separated in the graph. Worse, it has happened that two such vertices were identified when one lay at the edge of a cluster and the other was clearly a bridge. To address this, we only compare the persistence barcodes of vertices which are within a certain distance of one another. This has the added effect of reducing the number of Wasserstein distance calculations needed, which can be quite costly (see Discussion).

2.2. **Identifying Initial Cluster Points.** We now turn to the problem of finding vertices at the centres of clusters. The annulus test offers a partial solution but, in some cases, fails to identify any vertices if the cluster is densely connected (so that removing v does not disconnect the neighbourhood). To motivate the following method, we return to the previous discussion about the times at which large numbers of 1-cycles are born. From Figure 5, one would expect the vertices at the centres of clusters to be those which have a large portion of the 1-cycles born early in the shortest path filtration.

It is easy to compute the number of 1-cycles born at each stage of the filtration using the Euler characteristic method outlined earlier. We then mark vertices for which a large number of births occur sufficiently early as cluster points. Currently, "sufficiently early" is taken to mean that the largest portion of 1-cycles is born less than some fraction q of the way through the filtration. This test is implemented in such a way that q is as small as possible, meaning we start with the value $q = 0.05$ and increment q by 0.05 until any initial cluster points are found. We then fix q and stop searching for initial cluster points.
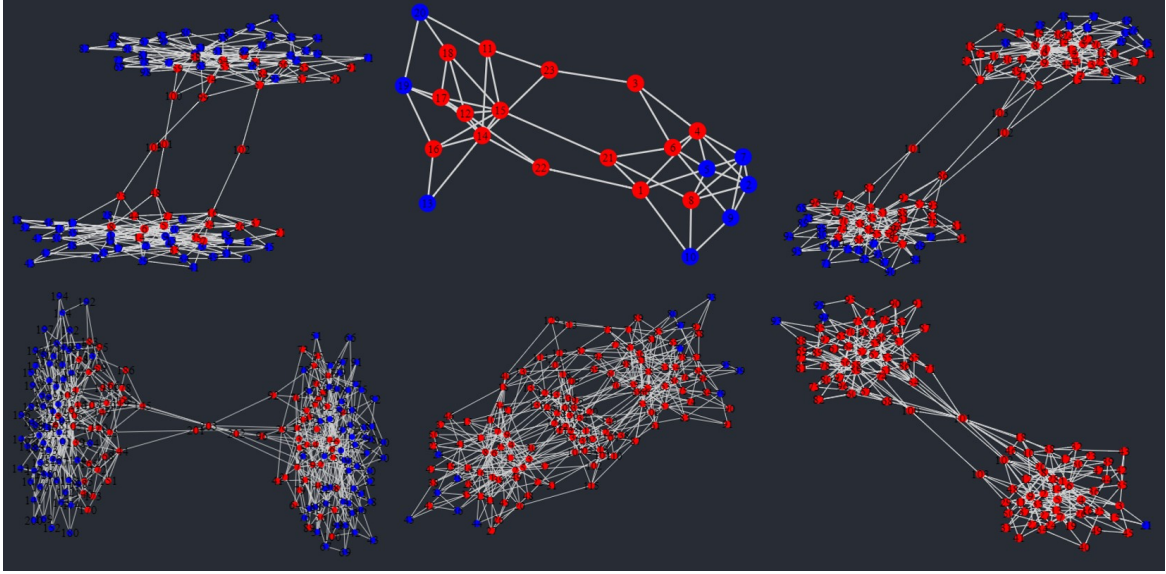
FIGURE 6. Examples of Using 1-cycle Birth Times to Identify Cluster Points

Figure 6 shows this method applied to various example graphs. In general, it will identify more vertices in densely connected clusters than in sparse clusters (compare the fourth and fifth graphs in Figure 6) simply because there are more 1-cycles in a densely connected cluster. The choice of q discussed above gives us an easy solution to the extremal problem but it has some drawbacks that should be discussed:

- The fewer initial cluster points are identified, the more Wasserstein distance computations need to be made and this is by far the most computationally expensive part of the algorithm.
- Suppose we have two distinct clusters, one densely connected and the other sparse. Because this test stops as soon as any cluster points have been identified, it is possible for cluster points in the dense cluster to be identified for small values of q while no cluster points are identified in the sparse one. This means that we have no vertex to compare the rest of the sparse cluster against, so we could fail to identify an entire cluster.

Both of these issues can be resolved by using a larger value of q. However, there is no longer an obvious way to make q dependent on the graph. This problem is still unresolved, but the scenario in the second point above happens very infrequently so it is not debilitating.

2.3. Graphs with More than Two Clusters.
We will now study how the algorithm performs on graphs with more than two clusters. Everything we have done thus far still holds in the general case and any assumptions (such as the connectedness of X) are still reasonable, for the same reasons as before.

The only real complication that arises with graphs with more than two clusters is that there are more opportunities for variation in the density of the clusters. This can cause the problem that, for a poor choice of q, we fail to identify any initial cluster points in some sparse cluster. We do not have a solution to this as of yet, so we will restrict ourselves to the class of graphs with clusters of similar densities. The majority of graphs used to model social networks fall into this class since communities tend to be densely connected.

As a partial solution should this problem occur, a user could manually identify some small set of initial cluster points and then proceed to use the Wasserstein distance to find the rest of the cluster.

Figure 7 shows the results of applying the 1-cycle birth test to graphs with more than two clusters. The results are generally in keeping with the two cluster case except for the exceptional case described above (not shown).
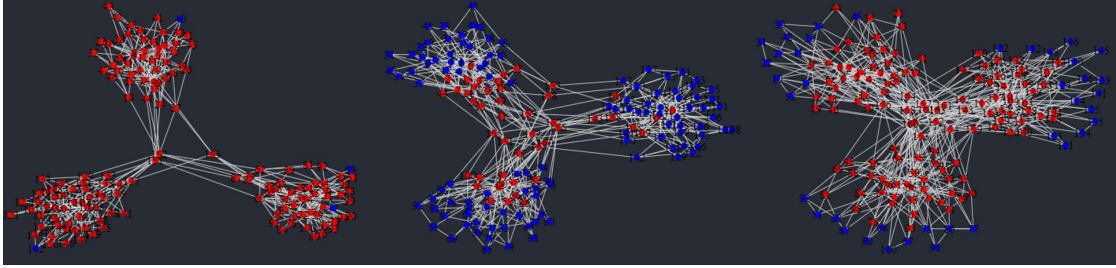

FIGURE 7. Initial Cluster Points Identified by 1-cycle Birth Test on Graphs with More Than Two Clusters

If one does not wish to work with graphs with more than two clusters, it is theoretically possible to reduce to the two cluster case (we have not implemented this). Start by identifying a set of initial cluster points and designate some neighbourhoods of these points as being potential clusters. We then choose any two of these potential clusters (if there are k potential clusters then there are $\binom{k}{2} = \frac{k(k-1)}{2}$ possible combinations) and include any vertices that lie between them as potential bridges. This gives rise to a connected subgraph which has at most two clusters, so we can run the algorithm from the two cluster case. We will not explore this approach as it would be more time-consuming than running the algorithm on the entire graph. Additionally, it still suffers from the problem of needing to identify initial cluster points to find the potential clusters.

2.4. **Choosing the Upper Bound on Wasserstein Distance.** There is one final part of the algorithm that we should address, which is choosing the upper bound on the Wasserstein distance between persistence barcodes. Assume that we have identified a set of initial cluster points and now wish to compare their persistence barcodes against their neighbours' to identify the rest of the cluster. Given two vertices v and w with corresponding 1-dimensional persistence barcodes A and B, we want to test if $W_{2,2}(A, B) \leq d$ for some upper bound d. As Figure 8 shows, the value of d greatly affects the accuracy of the algorithm.

$d = 5$ $d = 10$
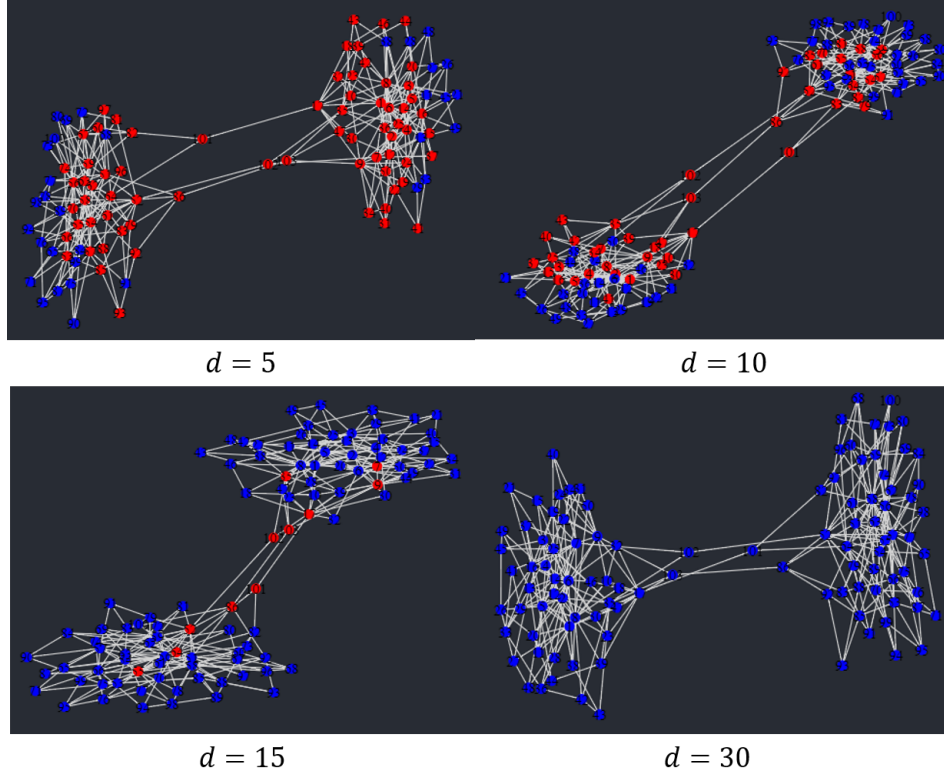
$d = 15$ $d = 30$

FIGURE 8. Algorithm Applied to a Fixed Graph Using Different Values of d

One can think of d as the radius of the cluster under the Wasserstein distance. If we already knew the set S of persistence barcodes corresponding to cluster points, we could use the standard definition of radius as $\frac{1}{2}\sup_{A,B\in S} W_{2,2}(A, B)$. Unfortunately, we are trying to use the radius to identify S so this isn't possible. Instead, we want a definition that doesn't depend on knowing S but is as close as possible to the above value.

This can be accomplished as follows: suppose we have already identified a cluster point c and a bridge b. The 1-dimensional persistence barcodes (call them C and B respectively) corresponding to these vertices should be the most separated in the graph. The value $W_{2,2}(C, B)$ is, in some sense, the "diameter" of the graph under the Wasserstein distance because it is the largest possible Wasserstein distance between any two persistence diagrams. We thus define d by

$$d = \frac{1}{2}W_{2,2}(C, B).$$

In general, this is a good value of d. For example, Figure 9 shows the same graph as in Figure 8 with $d = 18.76$, the value given by this formula.
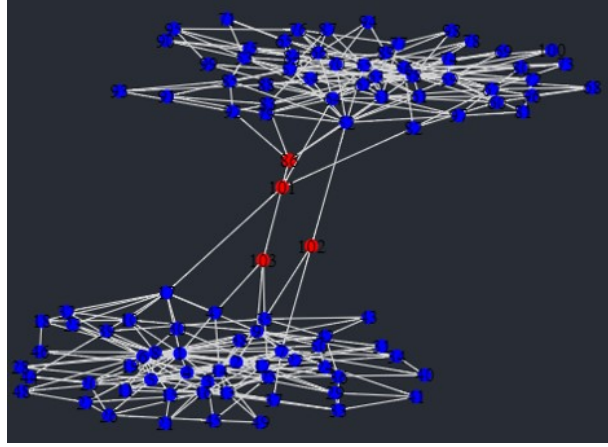
FIGURE 9. Graph with Value $d = 18.76$ Chosen Using Diameter Method

The obvious drawback to this method is that it requires knowing a bridge beforehand. To avoid this problem, we can compute the Wasserstein distance between *all* pairs of vertices and choose the maximum value to be the diameter. As we mentioned earlier, the computation of Wasserstein distance is easily the most computationally expensive part of the algorithm. If the graph has n vertices, using this method would means have to make $\frac{n(n-1)}{2}$ comparisons. However, if we store the results then we need not calculate them again once we know the value of d.

There is one other way to increase efficiency here, which is to exploit the fact that $W_{2,2}$ satisfies the Triangle Inequality. Suppose we are using the above method of comparing all pairs of vertices. We can keep a running maximum (call it M) to be used as the final value of the diameter. If we wish to take the Wasserstein distance between two diagrams A and C, and we have already computed the values $W_{2,2}(A, B)$ and $W_{2,2}(B, C)$ for some third diagram B, we can use the Triangle Inequality

$$W_{2,2}(A, C) \leq W_{2,2}(A, B) + W_{2,2}(B, C)$$

to test if it is necessary to compute $W_{2,2}(A, C)$. If the right-hand side of the above inequality is already less than M, there is no reason to compute $W_{2,2}(A, C)$ because it cannot be the new maximum. This significantly reduces the number of computations needed if we are in a cluster and have already compared a cluster point and a bridge.

3. DISCUSSION

We will now discuss the accuracy to the algorithm, as well as some potential avenues for further research. Let us quickly summarise the algorithm for easy reference:

(1) Identify initial cluster points by looking for vertices for which a large number of 1-cycles are born early in the shortest path filtration.
(2) Choose an appropriate value of d for use as the upper bound on the Wasserstein distance.
(3) By computing the Wasserstein distance between the 1-dimensional persistence barcodes, identify other vertices which are close under both the shortest path distance and the Wasserstein distance to also be cluster points. What remains are the bridges.

3.1. **Comparison with a Classical Clustering Method.** To justify the use of 1-cycle birth times to identify initial cluster points, let us compare this test to a more classical method for clustering. We define the Laplacian matrix L for a graph by the formula

$$L_{i,j} = \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

The matrix L is symmetric and positive-semidefinite, so its eigenvalues are all real and non-negative. An easy result (see [3]) is that the multiplicity of the eigenvalue $\lambda = 0$ is exactly the number of path-components in X. By our assumption that X is connected, this means that $\lambda = 0$ always has multiplicity 1.

The smallest non-zero eigenvalue of L is called the spectral gap. By the above, the spectral gap will always be the second eigenvalue when they are ordered. The associated eigenvector is called the Fiedler vector and it gives us a method for partitioning X into two connected clusters which we will now describe. Let s denote the spectral gap and f denote the Fiedler vector, so $Lf = sf$. Because eigenvectors are only defined up to scaling by a non-zero constant, we will have to check that any information we deduce from f is invariant under scaling.

For each vertex v_i, the sign of the ith entry of f tells us which of the two clusters v_i should belong to. That is, if the signs of f_i and f_j are different then v_i and v_j belong to different clusters. This property is clearly preserved under scaling because, although scaling by a negative constant value may change the sign of the entry, it will always preserve the property of having different signs.

Building on this, the magnitude of f_i directly relates to how close to the centre of a cluster v_i is. To account for scaling, we identify those vertices for which $|f_i|$ is greater than the mean as being initial cluster points. To show that this is invariant under scaling, suppose that X has n vertices and that $|f_i| > \frac{\sum_j |f_j|}{n}$. Then

$$|(cf)_i| = |c| \cdot |f_i|$$
$$> |c| \cdot \frac{\sum_j |f_j|}{n}$$
$$= \frac{\sum_j |c| \cdot |f_j|}{n}$$
$$= \frac{\sum_j |(cf)_j|}{n}$$

Thus this property is well-defined under scaling. To determine if our test using 1-cycle birth times produces reasonable results, let us compare it to the vertices identified by the Fiedler vector. Figure 10 shows the results of using the Fiedler vector applied to the same graphs as in Figure 6.
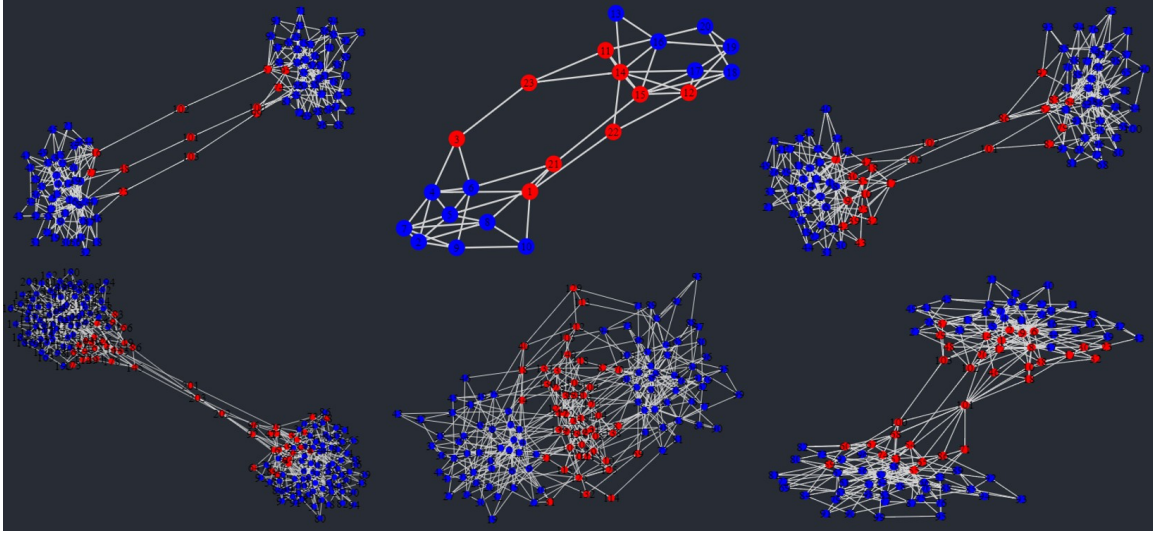
FIGURE 10. Example of Using Fiedler Vector to Identify Cluster Points

In general, the Fiedler vector identifies as many or more initial cluster points, as illustrated by comparing Figures 6 and 10. In fact, not once across twenty test graphs did the 1-cycle test identify a vertex as a cluster point that the Fiedler vector did not. Unfortunately, there is no generalisation of the Fiedler vector for graphs with more than two clusters; it is a special feature of *bipartitioning* a graph. However, that the 1-cycle test and the Fiedler vector agree in the two cluster case supports the accuracy of this method. Furthermore, if one chooses to reduce to the two cluster case, the Fiedler vector offers a method of refining the initial cluster points.

3.2. **Alternative Models for Social Networks.** Much of this algorithm has been built around the use of the shortest path filtration. However, for use in identifying diffusers of information in social networks, there is at least one other filtration worth considering. This is called complex contagion, which relies on the idea that a frequently shared message will build momentum; it will go viral. Using complex contagion to model a social network gives rise to a weighted directed graph, where the value of the edge (v_0, v_1) is the probability of v_0 sending a message to v_1. The fact that users who send more messages can be identified at the outset gives us a potential method for identifying initial cluster points.

For a given vertex v, we still have an associated filtration constructed in the same way as the shortest path filtration. The problem in using this filtration lies in the fact that the graph X is directed: homology theory is developed in the language of topological spaces which have no concept of direction. Thus much of the underlying theory of persistent homology (such as the existence of the persistence barcode) is lost. We can rectify this by assuming that the probability of a contacting b is the same as b contacting a, giving rise to a weighted, but undirected, graph that our algorithm can be applied to. The use of the complex contagion filtration is not one that we have tested, but it would certainly worth be considering in the future.

There are also methods for building higher-dimensional models for social networks (see [4]). Currently, the groups $H_0(X)$ and $H_1(X)$ are the only non-zero homology (in this paper, we defined them this way but it is actually a non-trivial result relying on the fact that X is what is known as a cellular complex). A higher-dimensional model

would allow for non-trivial higher homology groups, giving us more information about the geometric structure of X. The largest potential problem with this is that we could no longer use the Euler characteristic to compute homology. The formula for the Euler characteristic does generalise as an alternating sum of the ranks of homology groups, but we no longer have equally many equations as unknowns. There are other methods for computing persistence barcodes (one such is implemented in Julia via Eirene) but these tend to scale linearly or quadratically, as opposed to being constant time. On a graph with many vertices, this represents a significant decrease in efficiency. Should a higher-dimensional model be used, making Wasserstein distance computations across multiple dimensions would likely refine the ability to find other cluster points given an initial set, because a cluster could be considered as a higher-dimensional object while a bridge would generally just be 1-dimensional.

References

[1] A. Hatcher (2002). *Algebraic Topology.* Cambridge University Press. Available at **https://pi.math.cornell.edu/ hatcher/AT/AT.pdf**.

[2] M. Kerber, D. Morozov, A. Nigmetov (2016). *Geometry Helps to Compare Persistence Diagrams.* Available at **https://arxiv.org/abs/1606.03357**.

[3] B. Slininger. *Fiedler's Theory of Spectral Graph Partitioning.* Available at **http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.592.1730&rep=rep1&type=pdf**.

[4] D. Spivak (2009). *Higher-dimensional models of networks.* Available at **https://arxiv.org/abs/0909.4314**.

[5] S. Bhatia, B. Chatterjee, D. Nathani, M. Kaul (2018). *Understanding and Predicting Links in Graphs: A Persistent Homology Perspective.* Available at **https://arxiv.org/abs/1811.04049**.

AUSTRALIAN NATIONAL UNIVERSITY, MATHEMATICAL SCIENCES INSTITUTE