**TDA Presentation**

Over the summer, Kate and I have been collaborating with Marian-Andrei Rizoiu from the University of Technology Sydney to develop an algorithm capable of identifying diffusers of malicious information through social media. Today, I’ll be explaining this algorithm and trying to motivate its construction.

We model social networks as undirected graphs where vertices are users and edges represent communication between two users. In this model, a cluster of vertices represents a community in the social network, for example a Subreddit. To identify diffusers, then, we want to find vertices which aren’t members of any cluster because those are the users that don’t have an online presence except to spread information between communities. We’ll call such vertices bridges.

The idea of a bridge in a graph originated in the social sciences, so there unfortunately isn’t a rigorous definition. To address this, we define a partition of the vertices into bridges and cluster points, so identifying the bridges is equivalent to finding the cluster points. This is a much more common problem, and there exist methods like spectral clustering for clustering a graph. Our algorithm is another such clustering method.

To start with, we may assume without loss of generality that our graph is connected because we can just apply the algorithm in parallel to each of its path-components. For each vertex , we then have an associated filtration, called the shortest path filtration, where the stage is the induced subgraph of all vertices distance or less from . This filtration is finite precisely because is connected.

This gives us a way to associate a 1-dimensional persistence barcode to every vertex. We only consider the 1-dimensional barcode because the connectedness of means the 0-dimensional barcode will always just be a single entry, , and is a 1-dimensional simplicial complex so all higher homology is trivial. This means that the 1-dimensional barcode is the only non-trivial one, which is one of the reasons we intend to explore different filtrations in the future.

An important feature of persistence barcodes is the associated distance function called the Wasserstein distance. For those of you not familiar with it, the Wasserstein distance between two persistence barcodes and is given by . In Eirene, the main TDA package in Julia, the Wasserstein distance is computed using the Hungarian algorithm. This is at best cubic complexity in the number of vertices, and the computation of Wasserstein distance is the main bottleneck in computational efficiency.

Because the shortest path filtration is 1-dimensional, the death times in the 1-dimensional barcode will all be infinite. This means that we have an explicit formula for the Wasserstein distance between two such barcodes so long as they are sorted . Our algorithm, which computes these barcodes using the Euler characteristic, naturally sorts them so this formula represents a marked increase in computational efficiency. Now when I say a marked increase, I mean the difference between 10 minutes and 10 seconds, so this is an important optimization and it is one of the reasons we chose the shortest path filtration. Using the Wasserstein distance and the 1-dimensional persistence barcodes, we have another way of saying that two vertices look “close” which depends intrinsically on the topological structure of . We’ll soon see how this can be applied in our algorithm.

Consider Figure 2, which was generated using more classical clustering methods. Specifically, notice the red vertices in the upper right-hand corner. These vertices were misidentified as bridges, even though they clearly belong to the same cluster as their neighbours. This behavior was, surprisingly, quite common, but we never identified bridges as cluster points; we only had false positives. This is important because it means that we can simply refine our search once we’d made an initial pass, now with additional information about which nearby vertices are cluster points.

It is here that the Wasserstein distance becomes useful . Suppose we have two vertices, one which has been identified as a cluster point and the other which has not yet been categorized. If the Wasserstein distance between the 1-dimensional persistence barcodes corresponding to these vertices is small enough, meaning less than or equal to some value , these vertices should belong to the same cluster and so the other vertex should also be considered a cluster point . The value of that upper bound greatly affects the accuracy of the algorithm, which you can see from Figure 3. The choice of is still an unresolved problem which we intend to continue exploring in the future.

Now, using the Wasserstein distance reduces our problem to an extremal one; we need only identify the vertices at the centres of clusters and then we use the Wasserstein distance to identify the nearby vertices which are members of the same clusters . To motivate the method for identifying initial cluster points, consider the difference between the shortest path filtration for a bridge and a cluster point. For a cluster point in an ideal graph, there are a large number of 1-cycles born early in the filtration because of the interconnectedness of a cluster. In contrast, if we look at the shortest path filtration corresponding to a bridge, there are comparably few 1-cycles born early in the filtration and then there is a large “jump” as the neighbourhood grows large enough to include the clusters . Figure 4 shows column plots of the number of 1-cycles born at each stage of the shortest path filtration for a typical cluster point and bridge. As expected, the “jump”, the time at which a large number of 1-cycles were born, occurred earlier in the filtration for the cluster point than for the bridge, and the number of 1-cycles born at the jump was much larger for the bridge.

While this method is not refined enough to distinguish between a bridge and a vertex at the edge of a cluster, it does give us a method for identifying vertices at the centre of clusters: count the number of 1-cycles born at each stage of the filtration and identify vertices for which the “jump” occurs early enough. Our algorithm is implemented so that early means that the jump occurs less than some fraction of the way through the filtration. To choose that fraction , we start with and increment it by until any vertices at all are identified as initial cluster points. Then we stop. This means that we have identified as few initial cluster points as possible. We chose this implementation because, as I said earlier, this method is not refined enough to distinguish between a bridge and a cluster point that doesn’t sit at the centre of a cluster. Taking larger values of occasionally meant that we would incorrectly identify a bridge as a cluster point and there’s no way to fix this misidentification. We can change a bridge to a cluster point, but we can’t go back. Here are some examples of the initial cluster points chosen by this method.

The downside of this choice of is that we must restrict ourselves to a particular class of graphs; all of the clusters in the graph must be of a similar density. To see why this is necessary, suppose we had two clusters, one of which was densely connected and the other of which was sparse. Any vertex at the centre of the dense cluster would have many more 1-cycles born early in the filtration. This means that for a small value of , we may identify initial cluster points in the dense cluster but none at all in the sparse one. When we then go to compute the Wasserstein distances, there are no vertices to compare against in the sparse cluster and so we could miss an entire cluster. Fortunately, communities in social networks tend to be densely connected, so this isn’t a ridiculous assumption.

And that essentially summarizes the algorithm: we construct the 1-dimensional persistence barcodes, identify the initial cluster points by using the 1-cycle birth time “jump” test, and then use the Wasserstein distance to find the rest of the clusters. To finish, these are examples of the algorithm applied to a range of graphs. As you can see, it’s fairly accurate under the assumptions I’ve detailed here, and since it’s approximately cubic complexity, it’s comparable to other clustering methods in terms of efficiency.