An Introduction to TILO/PRC

Jesse Johnson

When scientists, statisticians and business analysts try to understand large sets of data, they often want to group the data points into smaller sets, called *clusters*. Roughly speaking, a cluster should be chosen so that the points in each cluster are more closely related to each other than to the points outside the cluster. For example, the two-dimensional data in Figure 1 seems to be made of three clusters, which are circled. We discuss below an algorithm called TILO/PRC for finding clusters. This algorithm was inspired by a technique called *thin position* that has been used to solve related problems in knot theory and 3-dimensional topology.

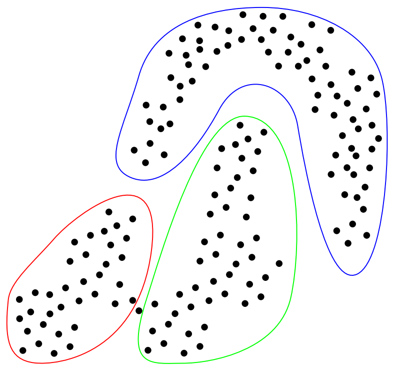

Figure 1: Clusters.

Most clustering algorithms fall into one of two categories: One type, including the popular algorithm K-means, treats the data as coordinates of points in a high dimensional Euclidean space and attempts to fit the points to a probability distribution in this space. Such algorithms can be fast, but do not always produce great results. For example, while they work well for spacial data, they are not as effective for data with many paramaters or data that does not have a nice geometric shape, such as the top cluster in Figure 1, which wraps around one of the other clusters. In general, they will perform poorly on data that does not fit the assumed model.

The second type of algorithm calculates the similarity between each pair of data points. There are different ways to define the similarity between two points, but the end result is that we can throw away the original coordinates and represent the relations between the data as a graph, i.e. a collection of points, called *vertices* (or *nodes* with lines called *edges* (or *links*) between them as in Figure 2.

The edges connect points whose similarity is above some threshold, and can also be marked with a number called a *weight* that records exactly how similar the two vertices are. For simplicity, we will assume below all the weights are 1. We will start by consid-

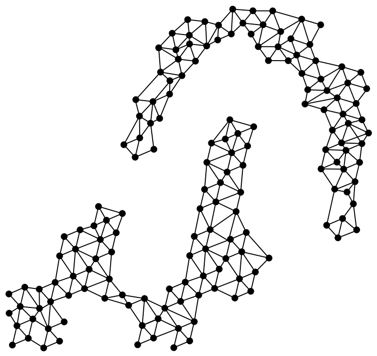

Figure 2: Encoding data as a graph.

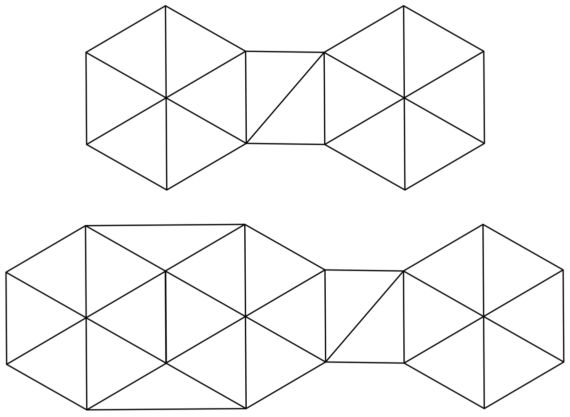

Figure 3: Two graphs with distinct clusters.

ering two very simple graphs, shown in Figure 3. For real data sets, the graphs will be much more complicated, but these examples will give us an idea of what's going on.

In the top graph we have 14 points of data, and the edges connect points that are closely related. The points are drawn in the plane as if they were spacial data, but in fact, they could come in any form, including data with hundreds or thousands of features/dimensions. The similarity measure and the graph representation allow us to represent the important aspects of the information relatively simply.

In the graph at the top of Figure 3, it appears that there are two clusters, each consisting of seven points, one on the left and one on the right. In particular, if we were to cut the graph in half between these two clusters, we would only have to cut along three edges. (If the edges had weights, we would consider the sum of the weights rather than just the number of edges.) On the other hand, if we were to try to cut it in half horizontally, for example, we would have to cut through a lot more edges (around eleven). We could also cut off one vertex by cutting only three edges, but this would not make a good cluster, since the goal is to find a large set of related points.

So, we might look at all possible ways to cut the points roughly in half and choose the one that cuts the fewest edges. The first problem with this approach is that computationally, it would be too time consuming to check every possibility for a very large set of points. The second problem is that the most appropriate cluster may be smaller or larger than half the set. For example in the lower graph in Figure 3, visual inspection suggests that the best place to cut would be between the 12 vertices on the left and the 7 vertices on the right, which cuts off roughly a third of the points.

The cut is "obvious" not because it splits the points evenly, but because it appears to be a bottleneck in the graph. We can make this precise by noting that if we were to slide the cut to the right or left across a single vertex, we would end up cutting more edges.

For example, in Figure 4, the original cut is shown in red. The two blue curves come from sliding the red cut across a single vertex. Each of the blue curves cuts four edges, while the original curve cuts only three. In fact, if we change the cut by any single vertex, it will cross more edges, so this seems like a good place to cut.

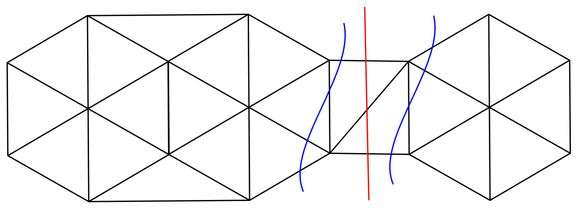

Figure 4: Moving the cut to the left or right makes it cross more edges.

In practice, the actual definition of a cluster is a bit more complicated than this, and you can read about the technicalities in [2]. For a large graph, it would still be computationally infeasible to look at all possible cuts in order to find clusters, so the TILO/PRC algorithm uses a more targeted approach.

In order to understand how the algorithm works, let's consider a real life bottleneck, or rather a bottle waist. Imagine that you have been given one of those old Coke bottles that has two bulges with a narrow waist in between them, and instructed to slide a rubber band over and across the bottle, then off on the other side. How would you do it?

Most likely, you wouldn't stretch the rubber band the long way across the height of the bottle, then pull it to from left and right, as illustrated on the left of Figure 5. Instead, it makes more sense to pull it over the mouth of the bottle, then down to the bottom, as on the right of the Figure.

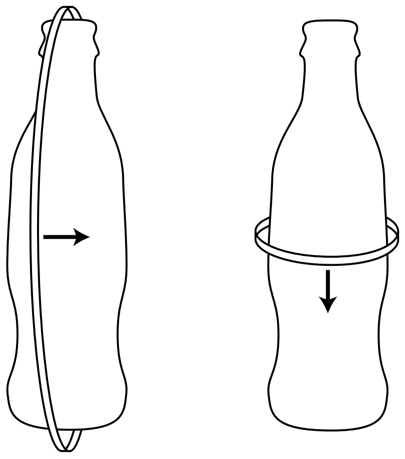

Figure 5: Sliding a rubber band over a coke bottle.

In fact, if you were to do it the second way, you would notice that the rubber band relaxes slightly in between the two bulges. In other words, the tension on the rubber band tells you when you've gotten to the bottle's waist. If the bottle were a data set, that's precisely where you would want to cut it to form clusters. So if you couldn't see the bottle, but could only measure the tension on the rubber band, you would still know that the bottle hads two bottlenecks.

The TILO/PRC algorithm looks for clusters by looking for the best way to "pull a rubber band over the data" (this is the TILO part), then measuring where the rubber band is not pulled as tight (this is the PRC part).

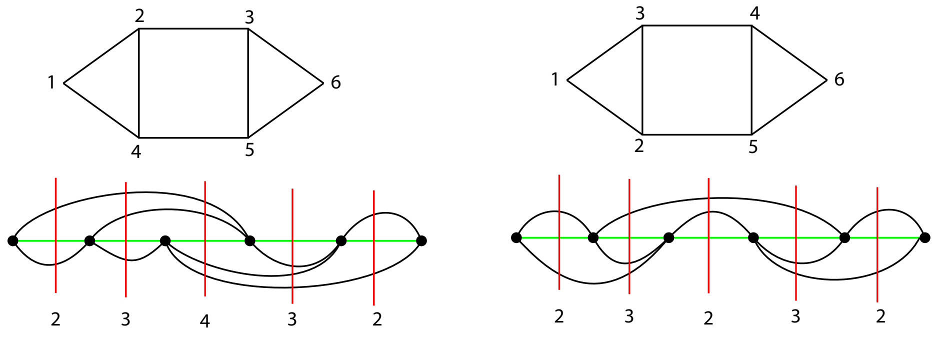

Figure 6: Two different ordering of the vertices.

More precisely, the algorithm begins with an ordering on the vertices/data points, i.e. we label them from 1 to N, where N is the number of data points. Imagine the points arranged along a horizontal line according to this order with the edges curving above and below the line. (For more complicated examples, these edges may have to cross each other.) This is shown for two orderings on a relatively simple graph in Figure 6.

On the left and on the right, the upper graph is labelled with the ordering. Below, each graph is drawn with its vertices along a horizontal line in the corresponding order. Between the vertices on the horizontal line, the red vertical lines indicate potential places to cut in order to find pinch clusters. Below each line is written the number of edges that the cut crosses.

Visual inspection of the graph suggests that there should be two clusters of three vertices, with the best cut probably being the vertical cut down the middle.

In the ordering on the left, this cut does not appear as one of the vertical red lines, and in fact the five suggested cuts are all fairly bad. However, the ordering on the right has the vertices in one cluster coming first followed by the vertices in the second cluster, so the middle cut along the vertical red line defines the clusters we want.

We can think of the sequence of five numbers given by these cuts - 2,3,2,3,2 - as the tension on the rubber band as we pull it across the data set. In this case, the middle number is smaller than the numbers immediately before and after it, so that's where the rubber band relaxes slightly. We call this a *local minimum.*

As with the coke bottle, even if we did not have such a nice picture of the graph, the local minimum would alert us to the fact that this cut probably defines a cluster. Of course, also like the coke bottle, this only works if we've chosen an efficient ordering.

In both orderings in Figure 6, vertices 1, 5 and 6 are the same. The difference is that the labels 2, 3, 4 have been switched around. Along the horizontal line, this corresponds to moving the second and third vertices in the first ordering to the right by one and moving the fourth vertex to the left, into the spot vacated by the second vertex. This is called a *shift* and turns the horizontal picture on the left into the horizontal picture on the right.

This shift changes the 4 edges defined by the middle vertical red line into 2. So the path of the rubber band on the right is “thinner” than the path on the left, hence the term “thin position”.

There is a relatively simple criteria for determining which shifts will make an ordering thinner. (See [2] for the precise definition of “thinner”.) The TILO portion (Topologically Intrinsic Lexicographic Ordering) of the TILO/PRC algorithm looks for this criteria in an initial ordering and repeatedly modifies the ordering to make it thinner. After some number of steps, it will find an ordering in which there are no shifts that make it thinner, and such an ordering is called *strongly irreducible*. For example, the ordering on the right in Figure 6 is strongly irreducible.

Once this ordering is found, there may be a large number of local minima, each defining a cut that will produce different clusters. The PRC (Pinch Ratio Clustering) algorithm chooses the best cuts by looking for minima where the tension is as small as possible relative to the tension at the peaks of the bulges before and after the waist [1].

Because this algorithm works by progressively improving an ordering on the vertices of the graph, the final result and the number of iterations depend on the initial ordering. This has two implications.

First, the algorithm can be run repeatedly to find different, possibly overlapping clusters in the data. The algorithm is not guaranteed to find a cluster (in fact, there are graphs where no pinch cluster exists) but experiments show that it is reasonably effective when run on a number of different random initial orderings [1]. When the algorithm does not find pinch clusters, the final ordering still suggests relatively thin cuts in the data, and can lend other types of insights into its structure.

Second, the algorithm can be used as post-processing to clean up the clusters suggested by a different algorithm. This has proved very effective in early tests of the algorithm [1].

As noted in the beginning, the ideas behind TILO/PRC come out of abstract three-dimensional topology. Therefore, this approach to clustering demonstrates an analogy between this abstract mathematics and very concrete problems in data analysis. By developing this analogy further, it should be possible to translate more ideas from the pure mathematics into flexible, efficient algorithms for analysing and visualizing data.

References

[1] Doug Heisterkamp and Jesse Johnson, *Pinch ratio clustering from a topologically intrinsic lexicographic ordering*, To appear in *Proceedings of the SIAM International Conference on Data Mining* (2013).

[2] Jesse Johnson, *Topological graph clustering with thin position*, preprint (2012), arXiv:1206.0771.