Yale

Extracting Progressions in Biological Data

Identifying Trajectories in Potential of Heat-diffusion for Affinity-based Transition (PHATE) Embedding

Ngan Vu

Advisor: Smita Krishnaswamy

A Senior Project in partial fulfillment for
the Combined Bachelor of Science and Master of Science
in Computer Science

Submitted May 02, 2019

YALE UNIVERSITY

Acknowledgements

I would like to express my profound thanks to Professor Smita Krishnaswamy for advising me throughout this project. I would also like to thank fellow lab members Scott Gigante, David van Dijk, and Kevin J. Moon, whose combined guidance was tremendously appreciated through many stages of this work. In the Krishnaswamy Lab, I have nurtured my love for research with incredible support behind me; the diligence and compassion of this team has given me the confidence and skills to continue my pursuits in the future.

Ngan Vu

Yale University

New Haven, May 2019

Abstract

As high-dimensional biological data becomes increasingly prevalent, there emerges a pressing need for visualization tools that reveal the underlying patterns and structures of data in intuitive forms. Potential of Heat-diffusion for Affinity-based Transition Embedding (PHATE, Moon et al. (2018)) is a visualization method that captures both local and global nonlinear structure by an information-geometry distance between data points. The advantage of PHATE over many other visualization methods is its ability to reveal branching structures that commonly exist in differentiation systems, such as stem cell development and disease trajectories. In this project, I use mathematical properties of graph-structured data sets, specifically the spectral properties of graph diffusion operators and the intrinsic dimensionality of signals on graphs, to develop and implement an algorithm that systematically detects and extracts such trajectories prior to PHATE-mediated visualization. I first determine end-points (points at one end of a trajectory) and branch-points (points around which data split in multiple directions); then, I associate all other points to trajectories defined by these two categories. I use this algorithm to successfully extract trajectories from three data sets of different natures: an artificial graph with branching structures, a simulated data set of single-cell RNA sequencing data, and a real embryoid body data set. I also discuss several less fruitful methods for trajectory detection, as well as future plans for improving the current algorithm. This trajectory detection project will eventually be distributed to the public as part of PHATE, allowing researchers to more successfully explore biological data sets and isolate data points of their research inquiries.

Keywords – PHATE, trajectories, end-points, branch-points

Contents

List of Figures

List of Tables

1 Introduction

The presence of large high-dimensional biological data set introduces a pressing need for tools that present data in a meaningful and intuitive way, so that researchers can understand and better discover the underlying structures of the data. Potential of Heat-diffusion for Affinity-based Transition Embedding (PHATE) (Moon et al., 2018) is a dimensionality reduction method specific for visualization. This means that PHATE generates a low-dimensional embedding, primarily in two or three dimensions, in a way such that structures like clusters and branches become apparent.

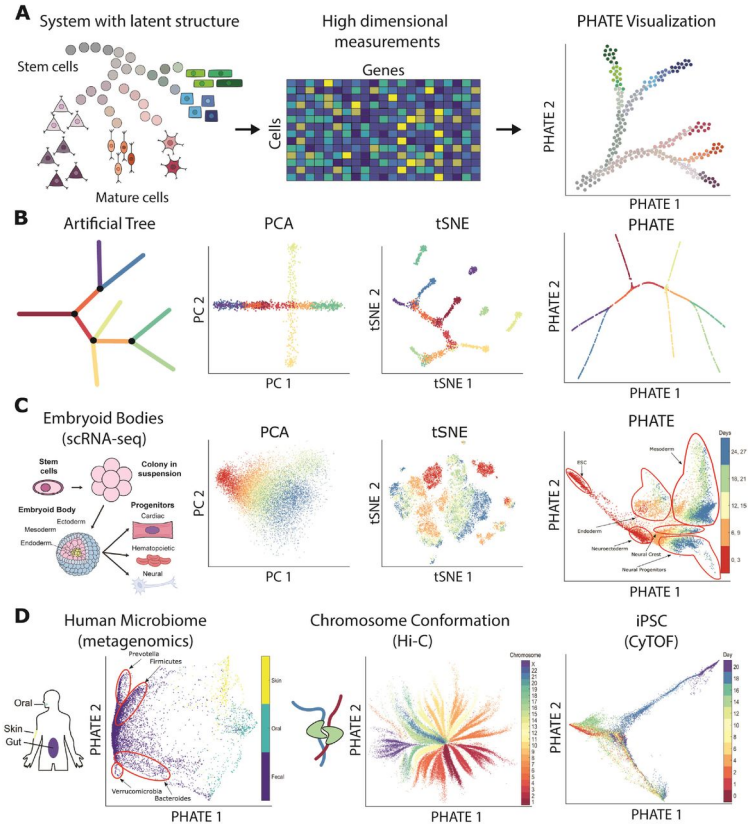

Figure 1.1: PHATE's ability to reveal structure in data (Moon et al., 2018)

One of the most popular feature requests from biology researchers who use PHATE is the ability to extract trajectories, which are usually visible in a PHATE embedding. Looking at a visualization, a human can usually points out existing trajectories. However, we would like to detect trajectories in a more systematic way, backed by mathematical rigor, for several reasons. First, we could detect trajectories that might not be very clear in the visualization and reject false trajectories. Second, we could extract the exact points that

belong to a certain trajectory of research interest, allowing researchers to have a closer looks at these points. This project automates the trajectory detection process as much as possible, while still allowing researchers to modify the suggested results to get the exact trajectories they want.

2 Background

2.1 Potential of Heat-diffusion for Affinity-based Transition Embedding

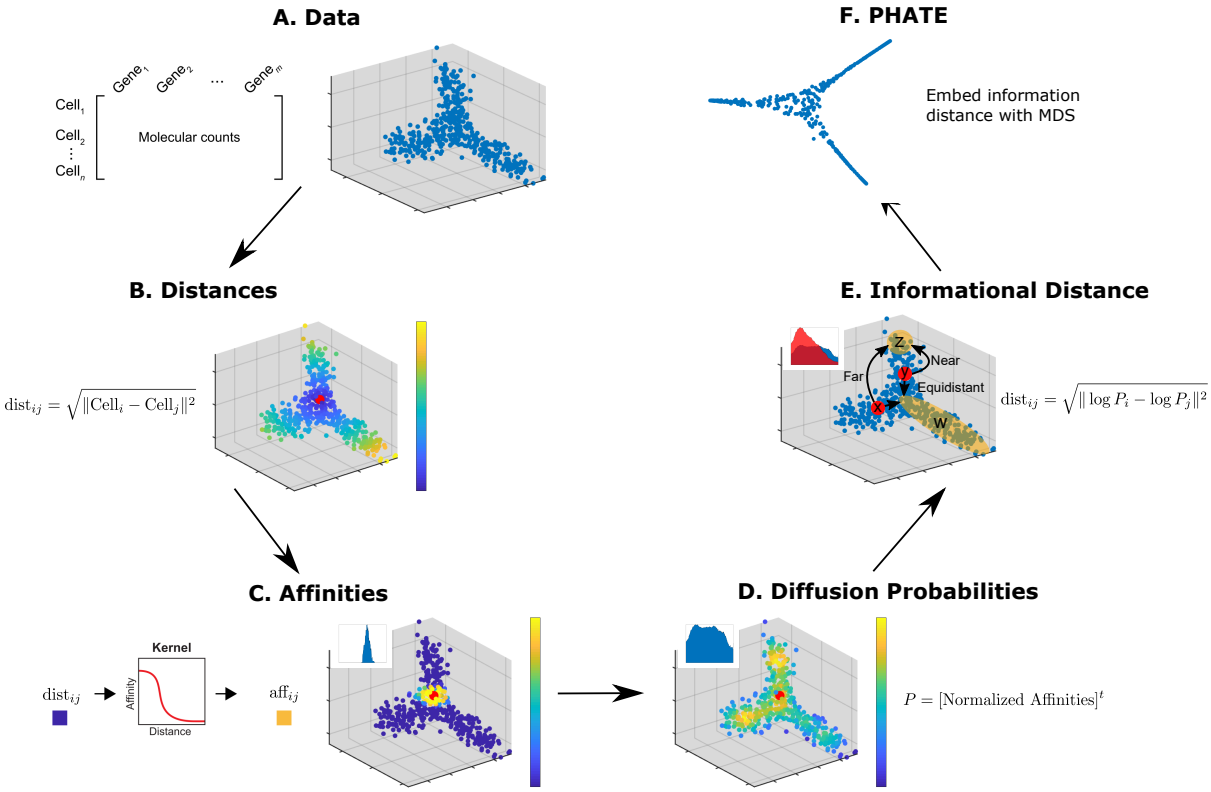

Figure 2.1: Graphical illustration of PHATE algorithm (Moon et al., 2018)

The above figure illustrates steps of the PHATE algorithm. (A) The input of PHATE is a two-dimensional matrix with cells (i.e. data points) as rows and genes (i.e. features) as columns. (B) The distance matrix contains the Euclidean distances between each pair of cells. Here, data points are colored by their Euclidean distance to the highlighted point. (C) Then, distances are converted to local affinities via a kernel function and normalized to a probability distribution, creating a Markov-normalized affinity matrix. Here, data points are colored by the probability of transitioning from the highlighted point in a single-step random walk. (D) Afterwards, the normalized affinity matrix is exponentiated, effectively denoise the data and learn long-range relationships between

points. The probability of transitioning over long distance is called the diffusion probability. Here, data points are colored by the probability of transitioning from the highlighted point in a t-step random walk. (E) Then, an informational distance, such as the potential distance, that measures the dissimilarity between the diffused probabilities is computed. The informational distance is better suited for computing differences between probabilities than the Euclidean distance. (F) Finally, the informational distances are embedded into low dimensions using Multidimensional Scaling (MDS), resulting in the final PHATE embedding.

Algorithm 1 The PHATE algorithm (Moon et al., 2018)

Input: Data matrix X, neighborhood size k, locality scale α, desired embedding dimension m (usually 2 or 3 for visualization)

Output: The PHATE embedding Y_m

1: $D \leftarrow$ compute pairwise distance matrix from X
2: Compute the k-nearest neighbor distance $\varepsilon_k(x)$ for each column x of X
3: $K_{k,\alpha} \leftarrow$ compute local affinity matrix from D and ε_k (see Eq. .1)
4: $P \leftarrow$ normalize $K_{k,\alpha}$ to form a Markov transition matrix (diffusion operator; see Eq. .2)
5: $t \leftarrow$ compute time scale via Von Neumann Entropy (see Eq. .3)
6: Diffuse P for t time steps to obtain P^t
7: Compute potential representations: $U_t \leftarrow -\log(P^t)$
8: $\mathfrak{V}^t \leftarrow$ compute potential distance matrix from U_t (see Eq. .4)
9: $Y_{class} \leftarrow$ apply classical MDS to $\mathfrak{V}^t$
10: $Y_m \leftarrow$ apply metric MDS to $\mathfrak{V}^t$ with Y_{class} as an initialization =0

One of the main advantages of PHATE is its faithful representation of the underlying trajectories, unlike many other dimensionality reduction and visualization methods. Therefore, it wouuld be useful to systematically extract and show these trajectories to researchers in the final PHATE visualization.

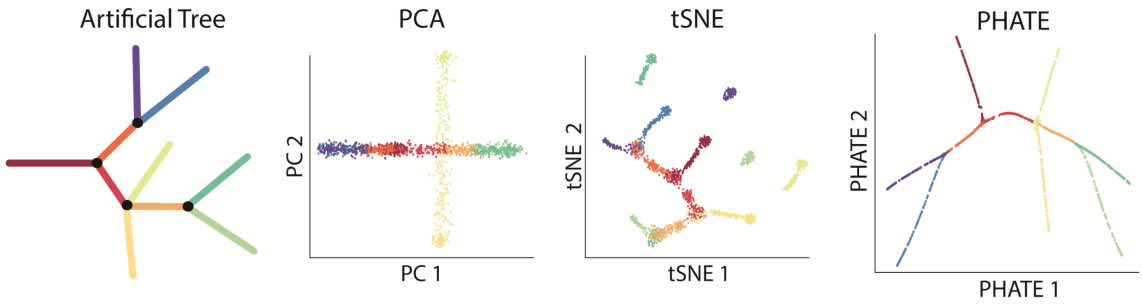

Figure 2.2: PHATE's ability to visualize trajectories (Moon et al., 2018)

2.2 Diffusion Maps

Diffusion Maps (Coifman and Lafon, 2006) are a classic non-linear dimensionality reduction method. The algorithm predates and partly resembles PHATE. The main difference between PHATE and Diffusion Maps is that PHATE is better at representing all information about the data in two dimensions or three dimensions, while Diffusion Maps require more dimensions to represent the same amount of information. In other words, PHATE is better for visualization. However, Diffusion Maps are very useful when the data only needs to be analyzed and not necessarily visualized, as we are not restricted to just two or three dimensions. In this case, we can look at as many dimensions as needed, each of which contains information about one certain trajectory.

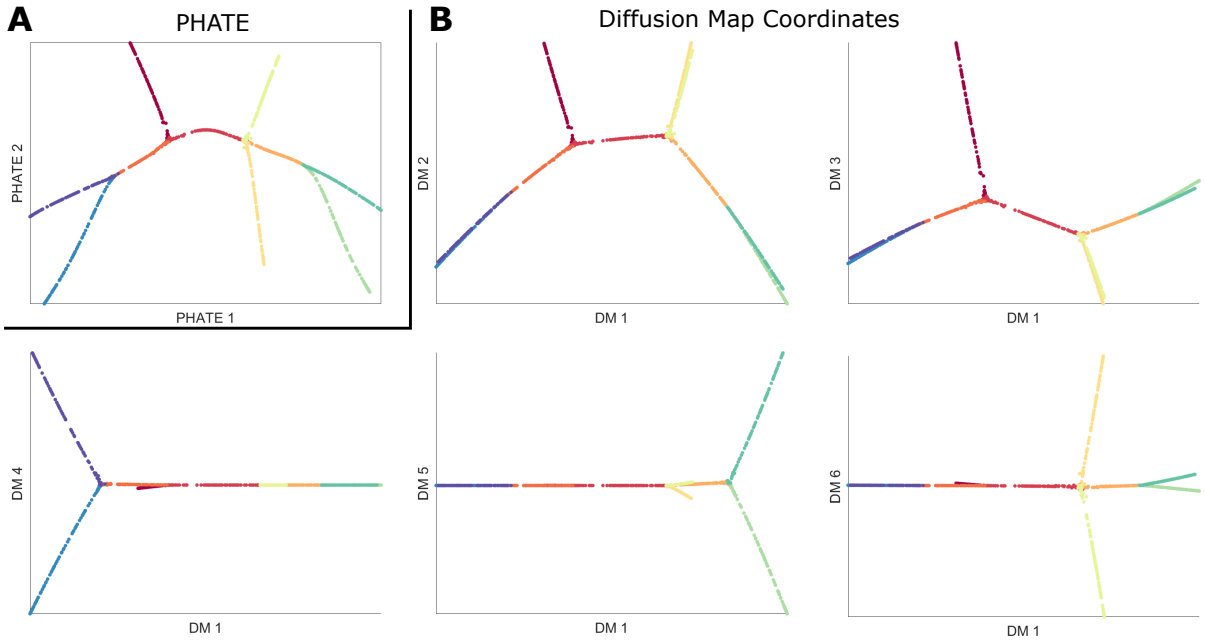

Figure 2.3: Compared to PHATE, Diffusion Maps require more dimensions to reveal all trajectories, but each dimension tells us more about one certain trajectory.(Moon et al., 2018)

3 Data

I work with 3 different kinds of data sets of various natures: one artificial (artifical PHATE tree), one simulating real-world (Splatter single-cell mRNA sequencing data), and one real-world (embryonic body data). The main category that I develop this method with is Splatter data, since it is easier to construct and test the method with different configurations of trajectories.

3.1 Splatter data

Splatter (Zappia et al., 2017) is an R package for the simple simulation of single-cell RNA sequencing data. Among other features, it is capable of generating data with customizable trajectories. Therefore, it is convenient to test trajectory detection on different data sets generated by Splatter.

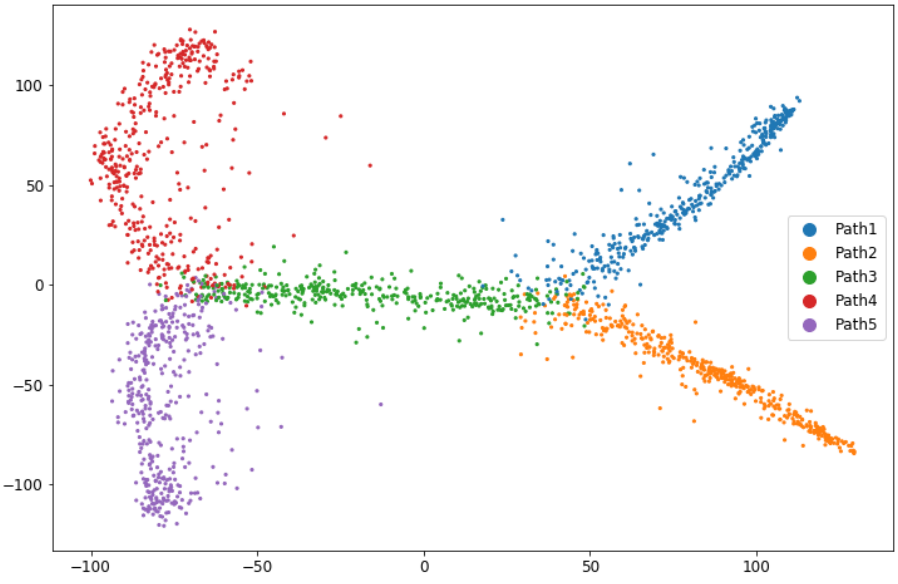

Figure 3.1: A data set generated by Splatter and visualized with PHATE

3.2 Artificial tree

Artificial PHATE tree is a data set designed to test the ability of PHATE to visualize trajectories; therefore, it is also useful to use this data set to test my trajectory detection method.

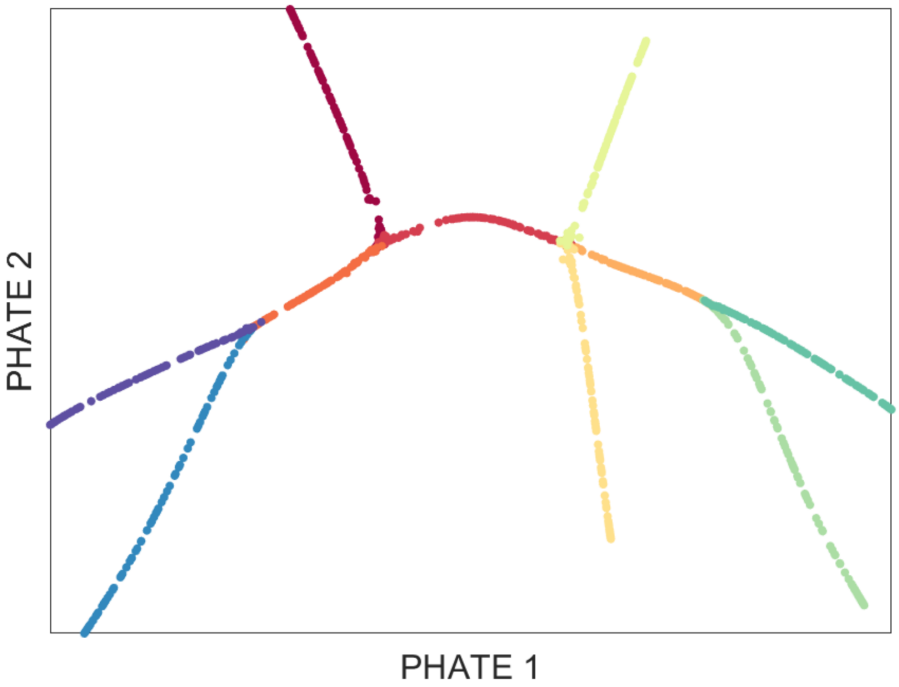

Figure 3.2: An artificial tree visualized with PHATE (Moon et al., 2018)

3.3 Embryonic body data

The embryonic body data set contains single cell RNA sequencing data from FACS sorted embryoid bodies. It was generated over 27 day time course using the indicated surface marker and used to demonstrate the data visualization capability of PHATE. This data set contains more data points with noise than the previous two and its underlying trajectories are less obvious. These properties make it a good real-world data set to test my method on.

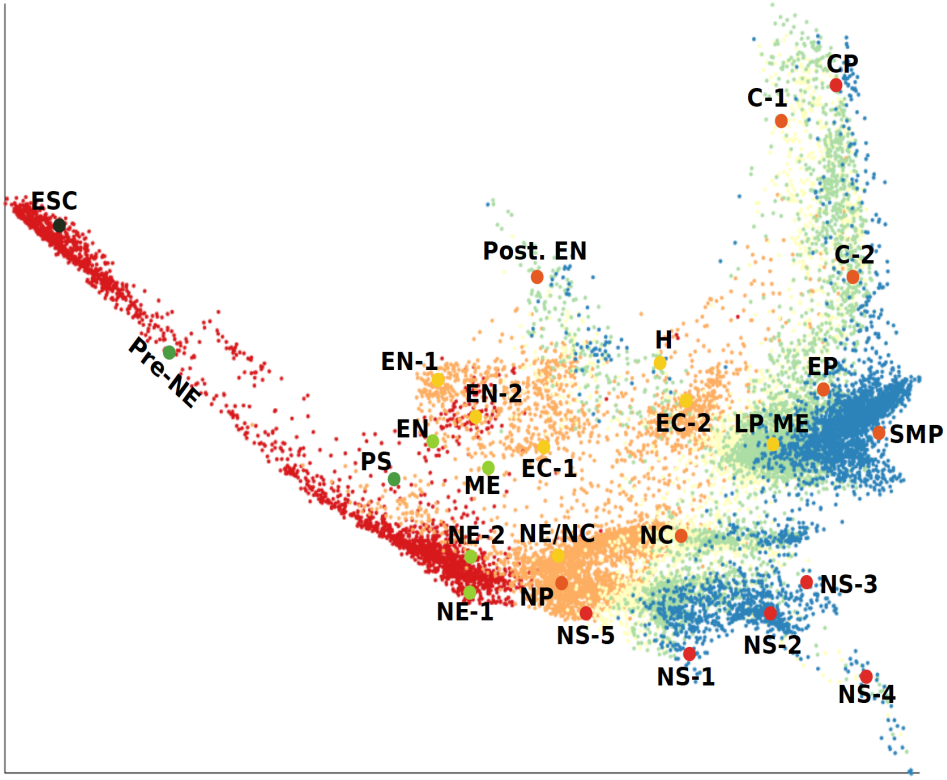

Figure 3.3: Embryonic body data visualized with PHATE

Table 3.1: Sizes of data sets

Data sets	Number of data points	Number of features
Splatter	2000	10000
Artificial Tree	3000	200
Embryoid Body	18691	17845

4 Methodology

Each of the steps for trajectory detection is detailed below.

4.1 Detecting end-points

I first detect end-points by using distinctness. The distinctness of a node is a measure of how unique a node is compared to the rest of the graph. In the context of graphs with trajectories, nodes that have the highest measure of distinctness tend to be those at the beginning or the end of trajectories. As shown in Cheng et al. (2016), I quantify this distinctness by considering the minima and the maxima of diffusion eigenvectors, which are also Diffusion Maps coordinates.

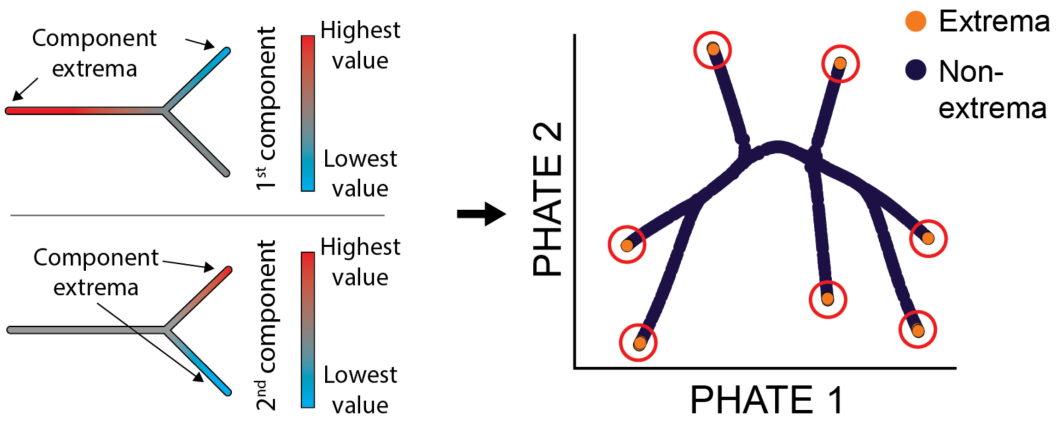

Figure 4.1: Detecting end-points with diffusion component extrema

An important question to consider is how many eigenvectors are necessary. Choosing too few eigenvectors will lead to missing end-points, while too many will lead to detecting fake end-points. In the figure below, we could see that when more eigenvectors are included, the extrema are not likely to be end-points. However, they usually are other meaningful points, such as branch-points.

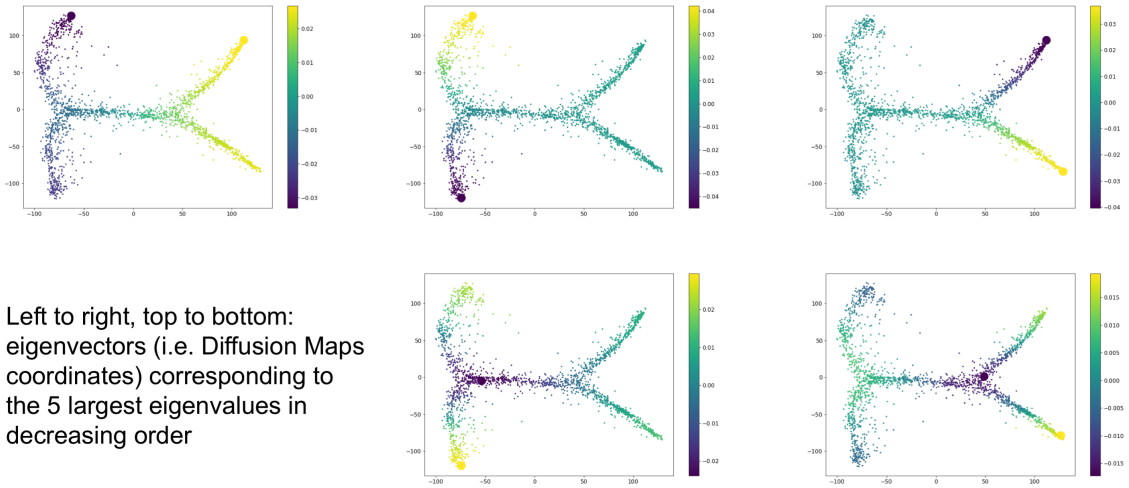

Figure 4.2: The first five diffusion coordinates colored on a PHATE embedding

In general, I choose the number of eigenvectors up to the point where eigenvalues drop significantly by looking at the step-wise changes in eigenvalues. For example, for this data set, eigenvalues drop significantly after the third eigenvalue. This means that we should only consider the first three diffusion coordinates. Note that this is consistent with the fact that the first three diffusion coordinates capture all the end points, as shown above.

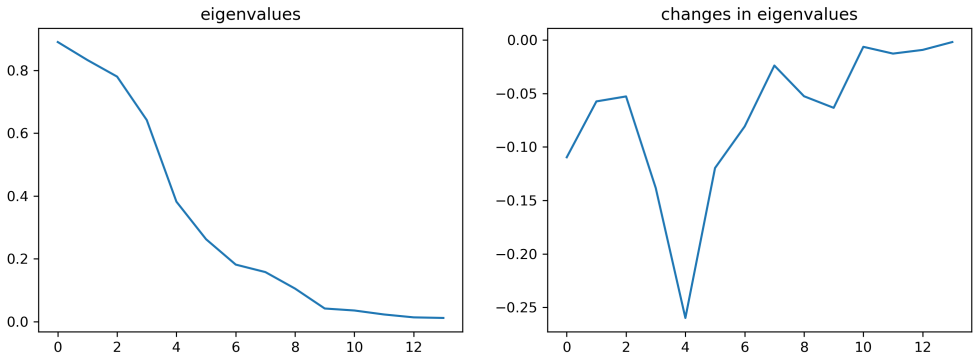

Figure 4.3: Eigenvalues and the changes in eigenvalues

It is usually better to have too many eigenvectors than too few, as it is possible to filter out fake end-points using intrinsic dimensionality: end-points should not have high intrinsic dimension.

4.2 Detecting trajectories corresponding to end-points

To determine the trajectory that starts or ends at and end-point, I first construct a Dirac vector with 1 at the index corresponding to the end-point and 0 everywhere else. Then, I diffuse this vector by left multiplying it with the diffusion operator for t steps. This is equivalent to simply looking at the column corresponding to the end-point of the t-exponentiated diffusion operator. This way, I can look at the diffusion from multiple end-points at once. For each column vector corresponding to an end-point, the significantly non-zero entries correspond to points on that end-point's trajectory. The higher the value of a point is, the closer that point is to the end-point.

After the diffusion of a Dirac vector, all points on the graph receive a value between 0 and 1 inclusively. An important question to consider is the threshold that determines if a point is on a trajectory or not, since we need to exclude points that receive minimal values from the end-point (which means they are too far away). This is a hyper-parameter that needs to be tuned or specified by users. Currently, I cut off points that are in the lowest quintile of the range of values in the diffused Dirac vector. This is an arbitrary approximation.

Another question is when to stop the diffusion. Stopping too early will make the trajectories shorter than they actually are, while stopping too late will make trajectories spill over to other parts of the graph. This question is addressed together with the method to detect branch-points below.

4.3 Detecting branch-points

Once we have a threshold for each trajectory, the branch-point is the point on the trajectory that is closest to the threshold. The lower a point's value on the trajectory is, the further it is from the end-point; therefore, branch-point should be the point with the lowest value possible on the trajectory. After each diffusion step, the trajectory gets more coverage and the branch-point changes. At some point, the branch-point will be the correct one, where data split or meet. To determine if the trajectory has reached the correct branch-point, I use intrinsic dimensionality.

The intrinsic dimension of a signal describes how many variables are needed to represent the signal. In the context of graphs with trajectories, the intrinsic dimension of a node is low for nodes in the middle of a trajectory (because the local neighborhoods of these nodes are one dimensional), while it is higher for nodes around where the trajectories split or meet (because the local neighborhoods of these nodes are multidimensional. There are various techniques for calculating the relative intrinsic dimensions of each nodes (also called point-wise intrinsic dimensions). The one I use for this project is Maximum Likelihood Estimation of Intrinsic Dimension (Levina and Bickel, 2005), which derives point-wise intrinsic dimensions by applying the principle of maximum likelihood to the distances between close neighbors.

The trajectory should continue to grow until the intrinsic dimension of its branch-point has saturated; this means that we have reached the area where trajectories split or meet, or we have reached an area of data where there are no trajectories.

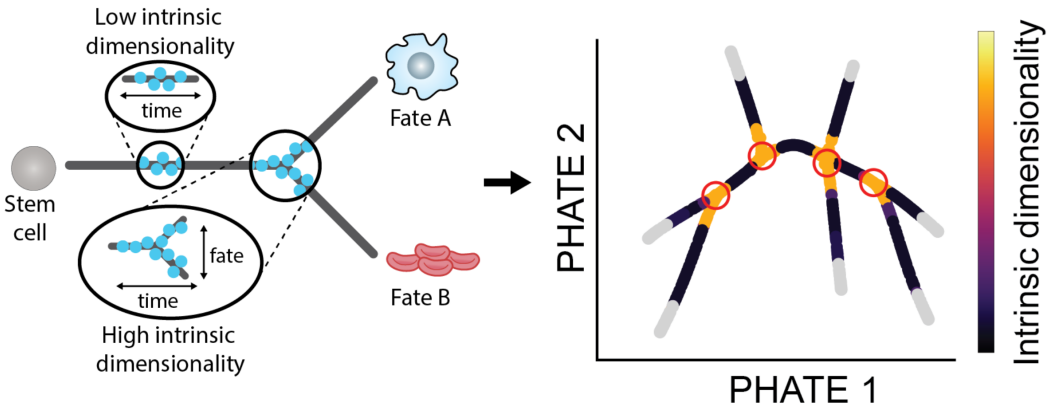

Figure 4.4: Detecting end-points with intrinsic dimensionality

4.4 Finalizing classes for each points

It is possible for points to belong to two different trajectories, especially branch-points. To visualize all points, we need to assign each point one single trajectory (i.e. class). If a point has more than one trajectory, it is assigned the trajectory corresponding to the end-point with higher eigenvalue, since this trajectory is more significant.

5 Analysis

5.1 Splatter data

I run this trajectory detection method on an unseen Splatter data set. The performance of each step is as expected.

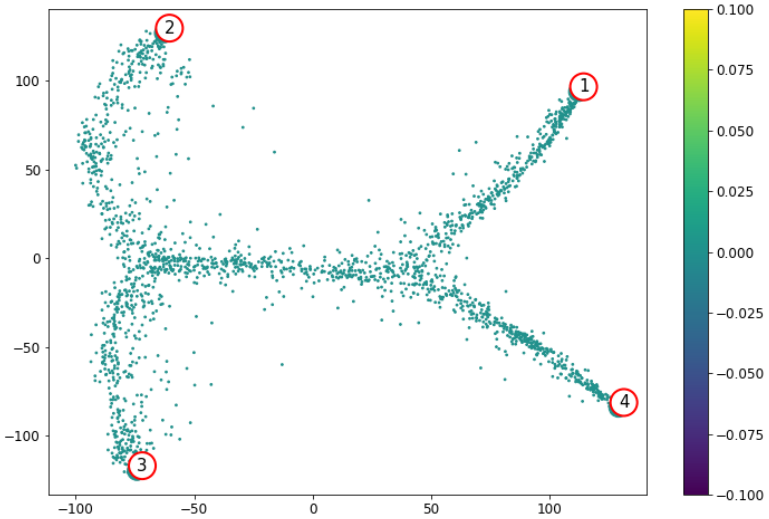

Figure 5.1: Detect end-points

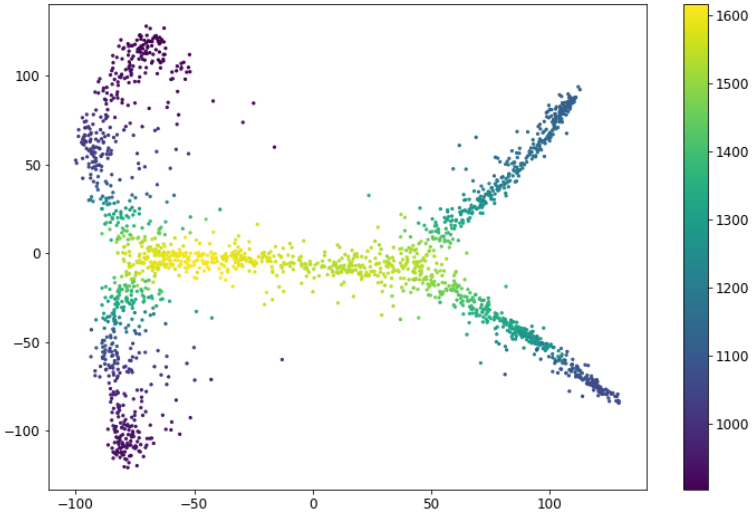

Figure 5.2: Calculate intrinsic dimension of each point

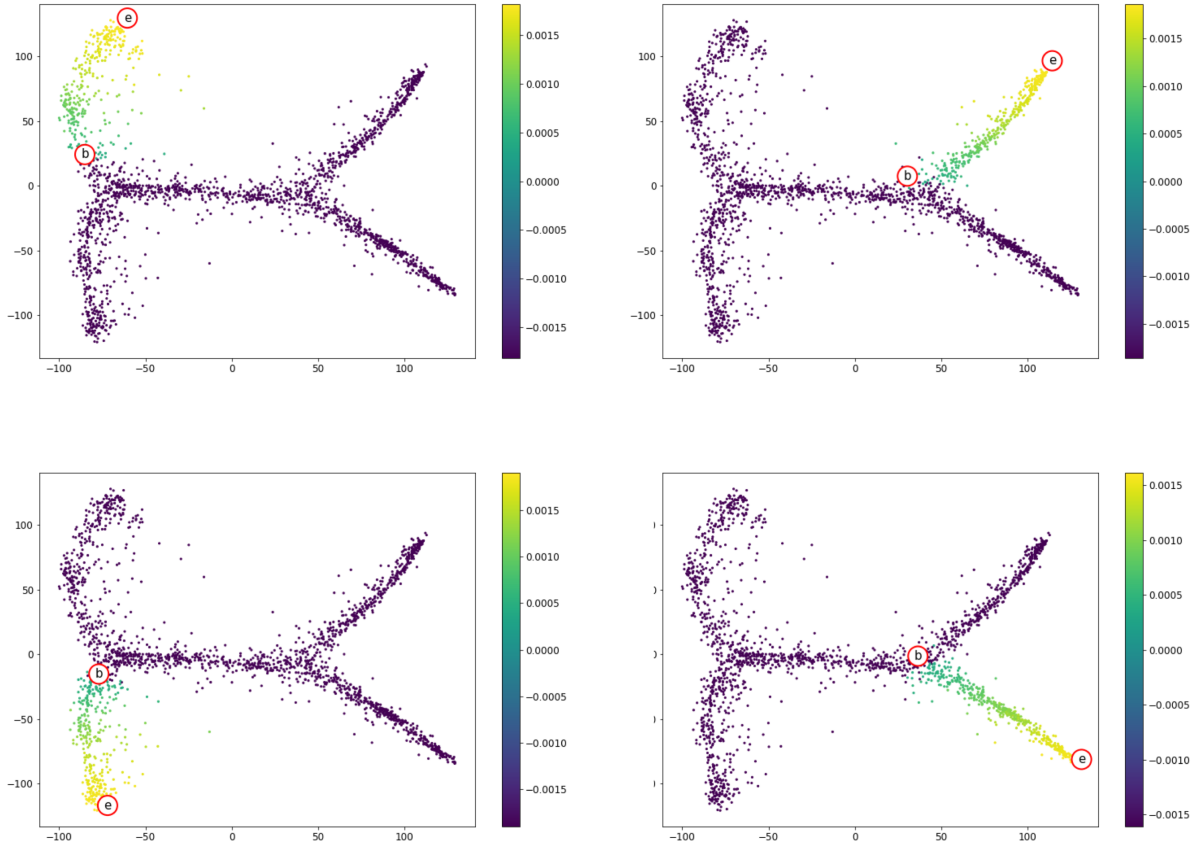

Figure 5.3: Detect trajectories and branch-points from end-points

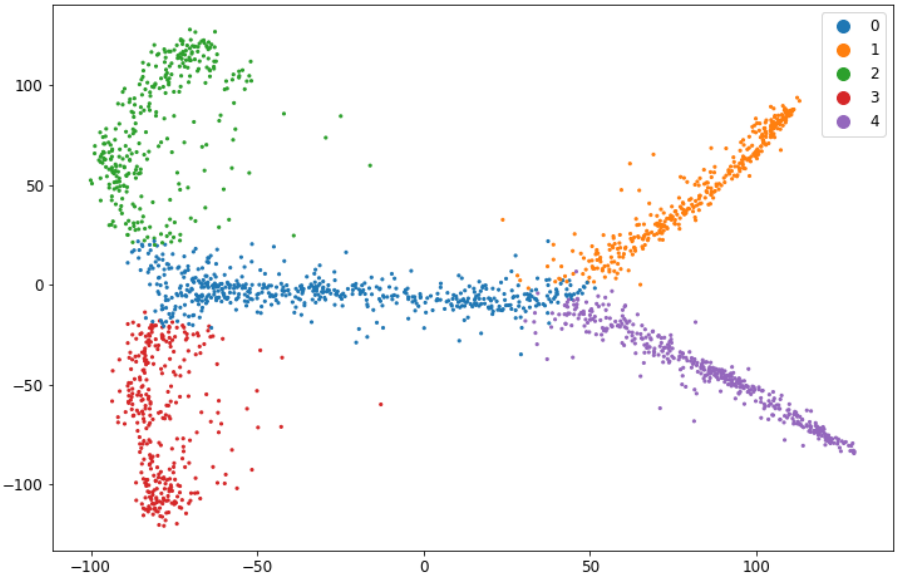

Figure 5.4: Final result

This final result is comparable to ground truth of the Splatter data.

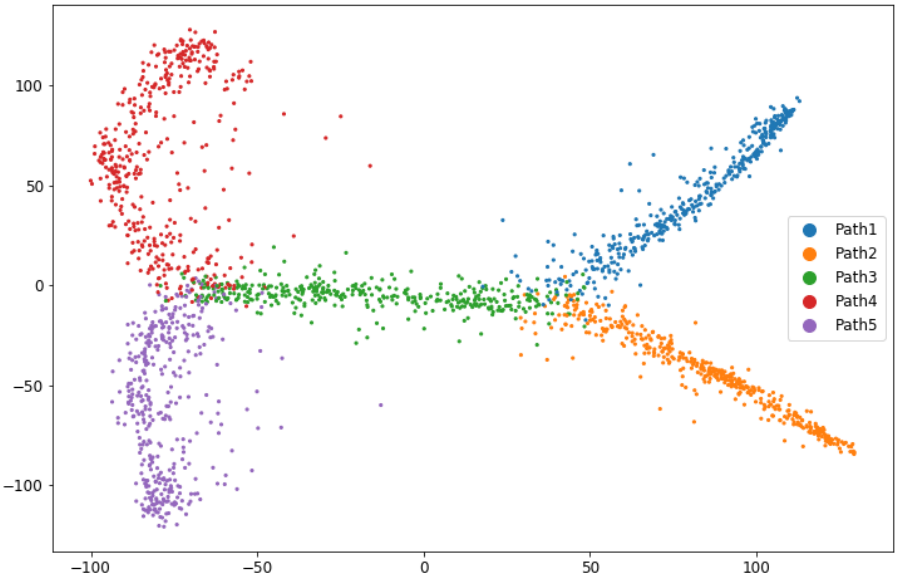

Figure 5.5: Ground truth

5.2 Artificial tree

This method is able to detect or branches on the artificial PHATE tree. However, it appears that some branches are cut off early, as shown below.

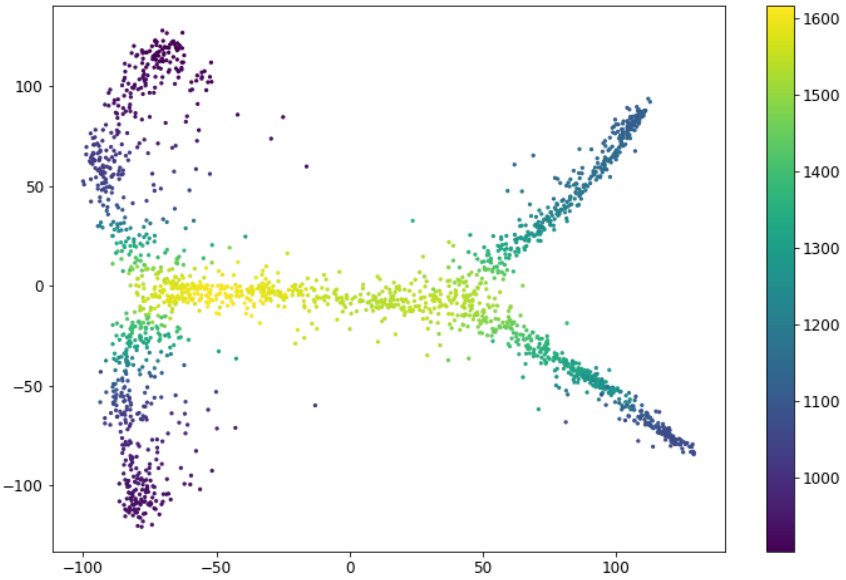

Figure 5.6: Final result

There are two solutions to this problem. The first one is to detect when intrinsic dimension plateaus more robustly. For example, it should be the case that the intrinsic dimension stops increasing in a few consecutive steps before we stop the diffusion. That way, a noisy slight decrease in intrinsic dimension should not cut off the branches too abruptly. The second solution, given that branch-points are quite visible in this case, is to allow users to manually make the trajectories continue to diffuse until they reach the desirable branch-points.

5.3 Embryonic body data

For embryonic body data, I was able to successfully detect the end-points. However, the diffusion process from end-point takes too long since there are too many points in this

data set, making the diffusion operator matrix huge. Therefore, I have not been able to detect trajectories from these end-points on this data set.

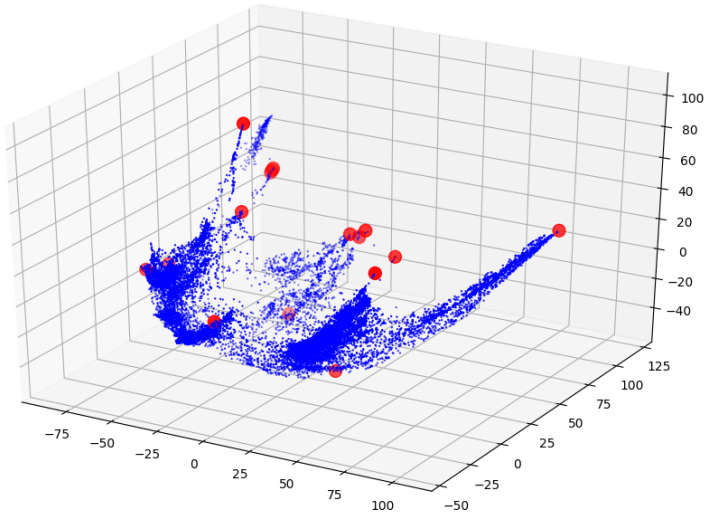

Figure 5.7: End-points of trajectories in embryonic body data set

In reality, PHATE actually does not use the full diffusion operator. Rather, it uses a trick discussed in Gigante et al. (2019): PHATE does all the computation on only a small subset of points representative of the data, then extracts the final embedding of these data points to other points in the data set. Using this trick, together with a few other approaches as mentioned in Discussion, I imagine that it will be possible to extend this method to much larger data sets like the embryonic body data set.

6 Discussion

6.1 Other attempted approaches

6.1.1 Using eigencentrality to detect branch-points

Eigenvector centrality, or eigencentrality, is an indicator of how important a node is in a graph. Each node in a graph is assigned a relative score such that a node with high eigenvector score is connected to many other high-scoring nodes. Eigencentrality and its variants have many applications in multiple domains. One of the most famous variants of eigencentrality is the Google's Page Rank algorithm (Page et al., 1998), which ranks web pages in the search engine's results.

Eigencentrality was one of the first ideas for detecting branch-points. I learned that eigencentrality is the first eigenvector of the diffusion operator of the graph, and that the first eigenvector can be calculated using Power Iteration method (Mises and Pollaczek-Geiringer, 1929). Unfortunately, high eigencentrality does not always indicate the presence of a branch-point. A node in the middle of a trajectory can still have high centrality if its trajectory is central to the graph, as illustrated below.

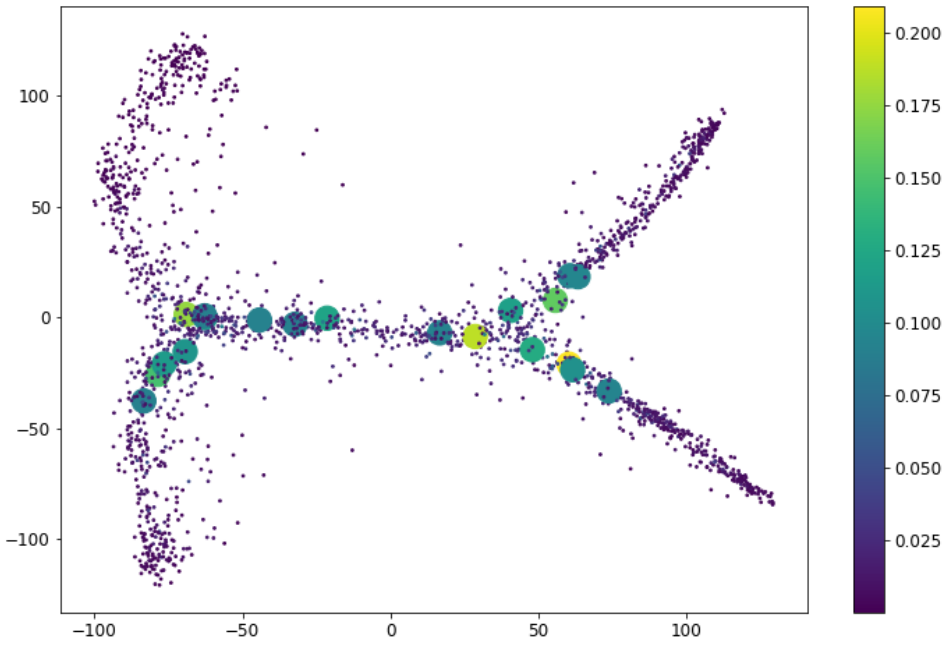

Figure 6.1: Centrality of points in a Splatter data set

6.1.2 Using intrinsic dimension to detect branch-points

Using only intrinsic dimensionality to classify branch-point is also a well motivated but not empirically successful approach. It is true that branch-points generally have higher intrinsic dimension compared to other points, but branch-points do not necessarily have the same intrinsic dimension. Therefore, we cannot simply choose points with highest intrinsic dimensionality as branch-points, since it is possible that these branch-points are skewed as illustrated below.

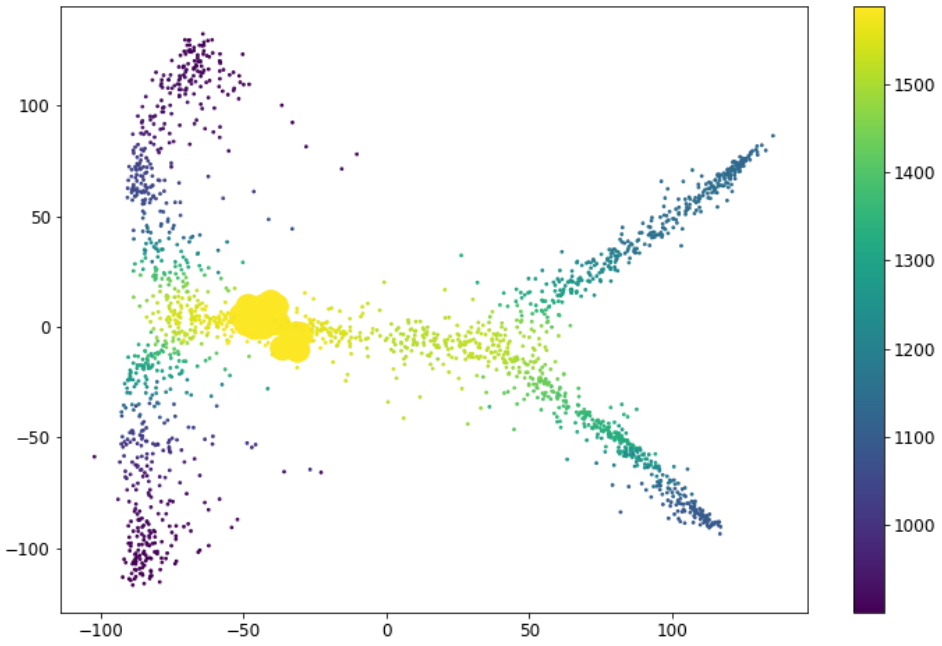

Figure 6.2: Intrinsic dimensions of points in a Splatter data set

6.1.3 Using abrupt changes in intrinsic dimension to detect branch-points

This approach is reasonable determine when a trajectory reaches a branch-point or the main body of data where there are no trajectories. However, the change in intrinsic dimension can be quite noisy. Besides, end-points also have large changes in intrinsic dimensions. These issues make this approach unsuitable for detection of branch-points.

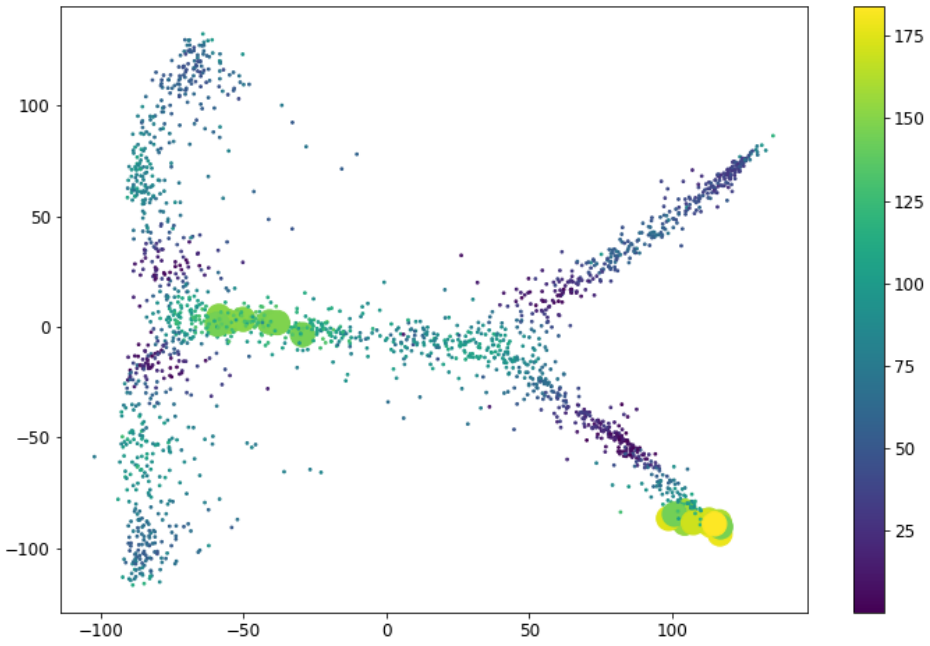

Figure 6.3: Change in intrinsic dimensions of points in a Splatter data set

6.1.4 Detecting extra distinct points, then classifying them as end-point or branch-points

As discussed in Methodology, sometimes when too many eigenvectors are used, fake end-points are detected. Usually, these fake end-points are branch-points, since branch-points are also more distinct than other points in the graph. Therefore, one possible way to get branch-points is to use more eigenvectors. Unfortunately, this approach does not work since points that are neither end-points nor branch-points are sometimes included. Even though it is easy to remove these points manually by looking at their location on the graph, it is difficult to remove them systematically using either centrality or intrinsic dimensionality, since they can be noisy and have surprisingly high centrality and/or intrinsic dimensionality.

6.1.5 Given branch-points and end-points, assigning other points to trajectories based on closest distance

There are two problems with this approach. First of all, Euclidean distance is not always representative of geodesic distance, the length of the shortest path between two points going through other points of the graph. Second of all, sometimes a point can be closest to an end-point and a branch-point that do not actually share a trajectory. In such case, we can try to find another end-point or another branch-point, but things get complicated quickly: Should we modify the branch-point? Or should we correct the end-point instead. There is no simple way to determine the answer to these questions.

6.2 Future plans

6.2.1 Optimize calculations of eigenvectors and eigenvalues

Currently, in order to find the eigenvectors and eigenvalues of the diffusion operator, I perform a full eigendecomposition. However, only the first few eigenvectors and eigenvalues are used. Therefore, it is better to use a faster eigendecomposition method that only calculates these eigenvectors and eigenvalues. One promising method is fast randomized singular value decomposition (SVD), developed in Halko et al. (2011), among others. This method only finds a specified small number of top singular values and singular vectors. Note that the singular values of a matrix are the square-roots of its eigenvalues, and the left and right singular vectors of a square matrix are its eigenvectors. Therefore, eigenvectors and eigenvalues can be quickly calculated with randomized SVD.

6.2.2 Optimize the diffusion process from each end-point

Currently, the branch corresponding to each end-point is detected via diffusion of a Dirac vector from that end-point. This diffusion process is repeated for each end-point. However, it would be more efficient to reuse the exponentiated diffusion matrix for all end-points to avoid expensive recalculations.

6.2.3 Order points on the trajectory

Based on the diffused Dirac vector from an end-point, we can rank all points on the graph based on the amount they received from the end-point after the diffusion process and use this metric to order points that belong to the trajectory leading to that end-point.

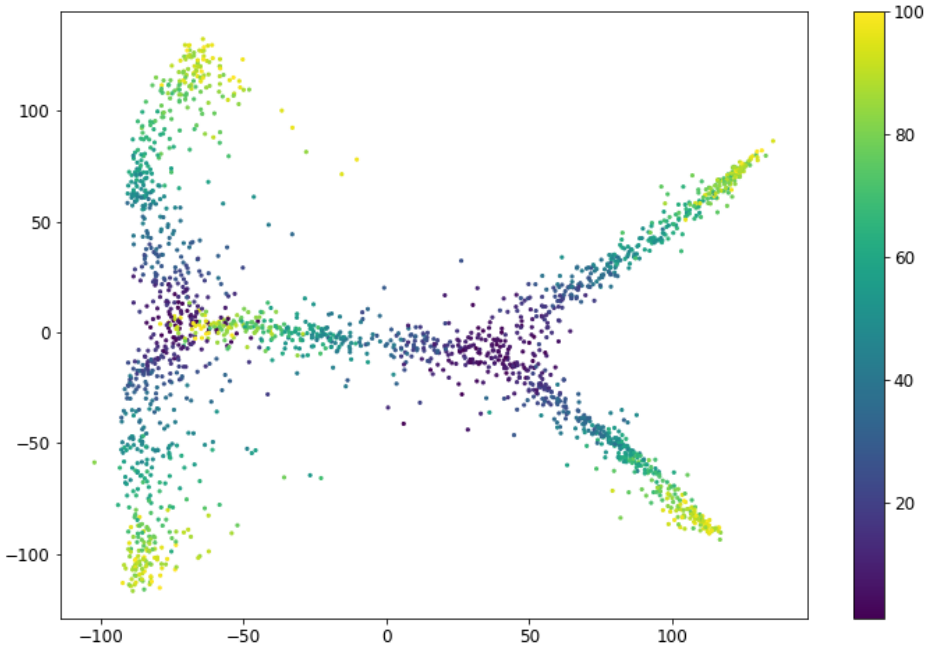

Figure 6.4: The order of points on paths generated by Splatter

6.2.4 Quantify correctness of method

Currently, performance of this trajectory detection method is only evaluated based on the visual results of PHATE. However, it is possible to evaluate how well this method perform when the trajectories are known. In Splatter data, for example, the path that each point belongs to as well as its relative order along the path, are provided. Therefore, it is possible to determine how my points are classified and ordered correctly.

6.2.5 Make use of PHATE landmarks

In large data sets, it is very costly to consider all data points. It is better to work with a small representative sets of data points, called landmarks, then extrapolate the information on these landmarks to the remaining points. This idea is described in Gigante et al. (2019) and is used in the implementation of PHATE. It helps speed up the algorithm tremendously.

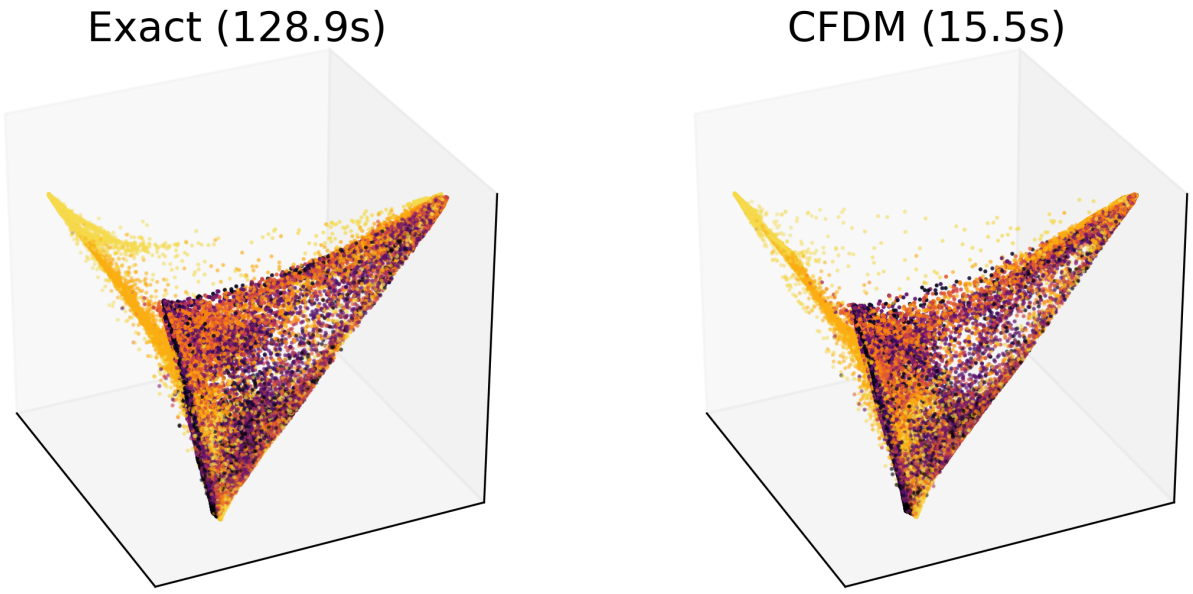

Figure 6.5: Speeding up diffusion methods with Compressed Diffusion. Here, CFDM is Compression-based Fast Diffusion Maps. (Gigante et al., 2019)

I speculate that it is also possible to use the same idea for trajectory detection. We can detect trajectories on landmarks first, then assign trajectories to the remaining points based on which landmarks they are closest to.

6.2.6 Experiment with other methods for estimating point-wise intrinsic dimension

While the technique proposed in Levina and Bickel (2005) works decently well, there are other techniques that I have not done extensive research into. Carter and III (2008) addresses high variance for high dimensions in intrinsic dimensionality estimation by adding adaptive neighborhood smoothing, reducing the variance and increase the accuracy

of the algorithm. Haro et al. (2000) addresses the case of noisy data by modeling distances between nodes with noise. These two methods can make the detection of branch-point for each trajectory more robust.

7 Conclusion

In this project, I develop a method to systematically extract trajectories from a data set before visualizing them with PHATE, by detecting end-points and branch-points using mathematical properties of graphs. I discuss approaches that worked and those that did not, as well as ideas for future developments. As trajectory detection is one of the most popular features requested for PHATE, I hope to make my code even more robust and release this project as part of PHATE to the public in the near future.

References

Carter, K. M. and III, A. O. H. (2008). Variance reduction with neighborhood smoothing for local intrinsic dimension estimation.

Cheng, X., Rachh, M., and Steinerberger, S. (2016). On the diffusion geometry of graph laplacians and applications.

Coifman, R. R. and Lafon, S. (2006). Diffusion maps. *Applied and Computational Harmonic Analysis*, 21(1):5 – 30. Special Issue: Diffusion Maps and Wavelets.

Gigante, S., III, J. S. S., Vu, N., van Dijk, D., Moon, K., Wolf, G., and Krishnaswamy, S. (2019). Compressed diffusion. *CoRR*, abs/1902.00033.

Halko, N., Martinsson, P. G., and Tropp, J. A. (2011). Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. *SIAM REV*, 53(2):2011.

Haro, G., Randal, G., Sapiro, G., Haro, G., Randall, G., and Sapiro, G. (2000). Translated poisson mixture model for stratification learning. *Int. J. Comput. Vision*.

Levina, E. and Bickel, P. J. (2005). Maximum likelihood estimation of intrinsic dimension. In Saul, L. K., Weiss, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 17*, pages 777–784. MIT Press.

Mises, R. V. and Pollaczek-Geiringer, H. (1929). Praktische Verfahren der Gleichungsauflösung . *Zeitschrift Angewandte Mathematik und Mechanik*, 9:58–77.

Moon, K. R., van Dijk, D., Wang, Z., Gigante, S., Burkhardt, D., Chen, W., van den Elzen, A., Hirn, M. J., Coifman, R. R., Ivanova, N. B., Wolf, G., and Krishnaswamy, S. (2018). Visualizing transitions and structure for biological data exploration.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1998). The pagerank citation ranking: Bringing order to the web.

Zappia, L., Phipson, B., and Oshlack, A. (2017). Splatter: simulation of single-cell rna sequencing data. *Genome Biology*.

Appendix

A1 Equations

A1.1 Adaptive Alpha-decaying Kernel

$$K_{k,\alpha}(x,y) = \frac{1}{2}\exp\left(-\left(\frac{\|x-y\|_2}{\varepsilon_k(x)}\right)^{\alpha}\right) + \frac{1}{2}\exp\left(-\left(\frac{\|x-y\|_2}{\varepsilon_k(y)}\right)^{\alpha}\right) \qquad x,y \in \mathcal{X} \quad (.1)$$

A1.2 Diffusion Operator

$$[P_\varepsilon]_{(x,y)} = \frac{k_\varepsilon(x,y)}{\nu_\varepsilon(x)}, \qquad x,y \in \mathcal{X}\,. \quad (.2)$$

A1.3 Von Neumann Entropy

$$H(t) = -\sum_{i=1}^{N}[\eta(t)]_i \log[\eta(t)]_i\,, \quad (.3)$$

A1.4 t-step Potential Distance Matrix

$$\mathfrak{V}^t(x,y) \triangleq \|U_x^t - U_y^t\|_2 \qquad x,y \in \mathcal{X} \quad (.4)$$