October 2021

Using Hyper-Dimensional Spanning Trees to Improve Structure Preservation During Dimensionality Reduction

Curtis Thomas Davis
University of South Florida

Using Hyper-Dimensional Spanning Trees to Improve Structure Preservation During Dimensionality Reduction

by

Curtis Thomas Davis

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Date of Approval:
October 18, 2021

Keywords: Data visualization, Hyper-dimensional relationships, Complex data,
Performance, Manifold learning

Dedication

To all the ones who led me here. Between my family and friends, newfound and old, I would not be where I am today without them.

Much appreciation goes to my major professor, Dr Paul Rosen. You guidance has been instrumental throughout my journey in school.

Acknowledgments

This material is based upon work partially supported by the National Science Foundation under Grant No. IIS-1845204. Any opinions, findings, and conclusions or recommendations expressed in this project are those of author(s) and do not necessarily reflect the views of the National Science Foundation.

I would like to give special thanks to Bei Wang and Ashley Suh for early conversations and ideas on this work.

Table of Contents

List of Tables

List of Figures

Abstract

Understanding relations in hyper-dimensional data is a prevalent problem, which is often approached by using dimensionality reduction. The structure preserved from the original data is often dependent on the type of dimension reduction algorithm used, and it can produce results that vary substantially from one another. Visualizing hyper-dimensional data helps to understand the data, but it presents a problem, as our visualizations rely upon two or three-dimensional displays. Current dimension reduction methods, used to reduce hyper-dimensional data to low-dimensional data, often produce results that fail to preserve the structure as the complexity of the data increases. This reduction in dimension means algorithms must make choices and disregard certain information. In this thesis, we improve upon existing dimension reduction methods through the use of hyper-dimensional spanning trees to preserve complex clustering relationships in the original data. We demonstrate the approach with an interactive program for our dimension reduction algorithm, which allows the user to fine-tune the outcome.

Chapter 1: Introduction

Understanding relations in hyper-dimensional data is a prevalent problem. An approach to solving this problem is utilizing dimensionality reduction, the transformation of hyper-dimensional data to lower-dimensional data. Dimension reduction has a broad series of applications, including improving methods in data mining (e.g., data preprocessing for effective machine learning), classification techniques, and in our case, data visualization. Dimension reduction is meant to reduce the dimensionality of data while preserving the more salient structures of the original data, intending to provide important insights of said original data. The structure preserved from the original data is often dependent on the type of dimension reduction algorithm used, and it can produce results that vary substantially from one another.

Visualizing hyper-dimensional data helps to understand the data, but it presents a problem, as our visualizations rely upon two or three-dimensional displays. Current dimension reduction methods used to reduce hyper-dimensional data to low-dimensional data, often produce results that fail to preserve the structure as the complexity of the data increases. This is not necessarily a fault of the algorithm, but rather due to the complexity of the problem to represent hyper-dimensional data in low dimensions. Reducing such hyper-dimensional data to a lower dimension, where we could better visualize it, means algorithms must make choices and disregard certain information instead of others. These choices, which result in a loss of certain information, opens the opportunity to harness additional information from the original hyper-dimensional data and apply it elsewhere in the process to lessen this limitation of current dimension reduction techniques.

In this thesis, we improve upon existing dimension reduction methods through the use of hyper-dimensional spanning trees to preserve additional information of the original data. We do this by creating an interactive program for our dimension reduction algorithm, which allows the user to fine-tune the outcome.

1.1 Motivation

Current dimension reduction techniques differ in their approach to preserve the overall structure of the hyper-dimensional data; dimension reduction techniques have a bias in the sense that it preserves certain qualities over others, such as longer distances over shorter ones or vice versa. Issues such as this can diminish the quality of the underlying structure preserved throughout the process. These existing algorithms also lack the ability for the user to fine-tune the results. Ideally, the user would be given useful parameters which allow them to guide the algorithm down a custom route to intuitively understand how their changes affect the outcome. Our algorithm capitalizes on the lack of involved parameters in these algorithms and gives the user parameters that can modify the steps of the algorithm and its resulting data, giving the user more insight to how their changes affect the dimension reduction of the hyper-dimensional data.

1.2 Contribution

This thesis consists of creating an interactive dimension reduction algorithm using hyper-dimensional spanning trees. Using these hyper-spanning trees, we demonstrate an algorithm that simultaneously betters the local and global structure preservation of existing reduction techniques. We also provide functionality with this algorithm that when used with our parameters, which are available to the user to edit, provides a result that will give new insights into the structural properties of the original data.

The dimension reduction techniques that we consider in this thesis operate by simplifying the hyper-dimensional dissimilarity, which is often the calculated distance between nodes in the original data, to a lower-dimensional version that we can better understand and manipulate. In essence, most dimension reduction algorithms begin with a pairwise dissimilarity, such as Euclidean distance, between each pair of n nodes, and calculates an $n \times n$ dissimilarity matrix to hold this dissimilarity information. When the techniques reduce the dimensions of the data, they use this dissimilarity data along with more specific calculations and information tailored to their methods, such as cluster and neighboring information, to preserve the underlying structure of the dataset. The problem we have noticed here is that to preserve more structure after the reduction process, we require more sophisticated structural information than these current methods make use of, in order to better the preservation of the structure.

Some approaches have tried to use other information to modify this dissimilarity matrix to improve graphs that represent visualized similarities of data. For example, using a Minimum Spanning Tree has been used to modify the distances of more similar nodes. However, in an $n \times n$ dissimilarity matrix, an MST could only modify $n - 1$ edges or pairwise relations within this graph [12]. This is a less than ideal situation, as there is not much information applied to the process to make a considerable contribution; these approaches only capture pairwise relations throughout the structures and overlook more complex relations involving more than two nodes.

Our algorithm remedies this issue by using the information of hyper-dimensional spanning trees to capture and utilize complex relations between the data while it performs the reduction process. Instead of only preserving one relation between two adjacent nodes, via a single edge in the dissimilarity matrix, we attempt to preserve the relationship of a larger amount of nodes through the use of hyper-edges. We use this additional information to better the preservation of local and global structures by modifying the dissimilarity matrices of current reduction techniques.

The specific contributions of this thesis are:

1. Providing a definition for a hyper-spanning tree, usable for extracting hyper-dimensional structure from hyper-dimensional point clouds.
2. Utilizing the information from hyper-dimensional spanning tree to improve both local and global structure preservation in current dimensionality-reduction techniques.
3. To provide the user with customizable parameters, which directly impact the resulting dimension-reduced graph, to provide insights of the data, and show the user how these parameters can be used to fine-tune the outcome.

Chapter 2: Background in Dimensionality Reduction

Dimensionality reduction can be used for many applications. One of the most important uses of dimensionality reduction is data visualization.

2.1 Hyper-dimensional Data

To visualize hyper-dimensional data, or high-dimensional data, one would have to visualize hyper-dimensional point clouds, which are graphs whose space is equal to or higher than four dimensions ($x \in \mathbb{R}^p$, where $p \geq 4$), such as a point cloud based on a dataset comprised of instances with some arbitrary number of dimensions. One example is a data set where the objects are flowers, known as the Iris dataset [8], and the dimensions are a few attributes of the flower (petal length, width, sepal length, etc.). These attributes act as the coordinates of each item.

The MNIST Data Set [17] is another a classic example of this, being a database of handwritten digit images comprised of over 60,000 training images and 10,000 testing images; this is a set of images where the number of dimensions is the product of the dimensions of the image (Width x Height = number of pixels in the image), and their coordinates are the pixel values. The conversion of such images to a coordinate vector can be seen in Figure 2.1.

2.2 Dimension Reduction Techniques

Dimension reduction aims to take a hyper-dimensional data set or point cloud and reduce the dimensions of the data. The intention of this thesis is that we want to reduce hyper-dimensional data to a lower-dimensional space while still retaining most of the in-

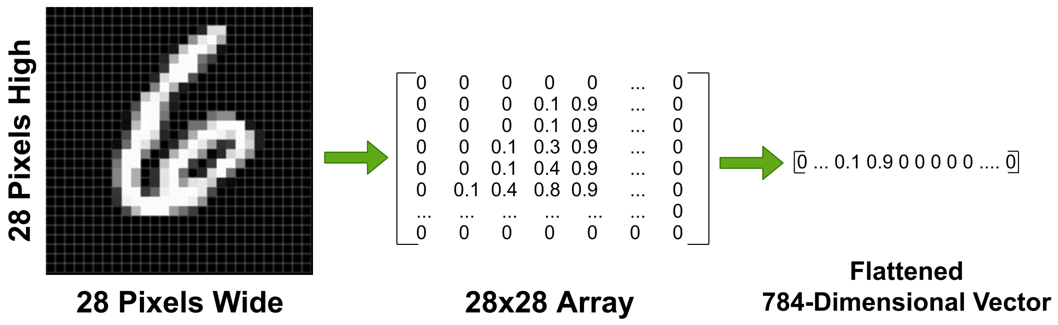


Figure 2.1: Hyper-dimensional images. An image can be represented by a two-dimensional array, where each cell is the value of each pixel. Here, we see an image from the MNIST dataset (left), where it has dimensions of 28 × 28. This is converted into a 28 × 28 array (middle), where the values of said array are the grayscale values of the image's pixels, on a scale from 0 being black to 1 being white. This can be further reduced to a flattened vector, where the length of the vector is the product of the dimensions of the image or array. This final vector represents the hyper-coordinates of the respective image in the hypergraph, where the image is a hyper-node.

formation and structure of the original data, $f : \mathbb{R}^p \to \mathbb{R}^2$, and to allow the data to be visualized.

Dimensionality reduction techniques use either a linear or non-linear method to map hyper-dimensional data to a lower-dimensional space; these techniques often have high-quality mapping results. However, they tend to lose global or local structure data during the process depending on the structures they are preserving or the complexity of the data being reduced. The idea behind structure preservation is that while a hyper-dimensional point cloud is mapped to the lower dimension, the original and resulting graphs are as isomorphic as possible to one another in their relations and form; we preserve structure because we want to retain as much of the original information and features that comprise the original dataset after dimension reduction has been performed. Uniform Manifold Approximation and Projection (UMAP) and t-distributed stochastic neighbor embedding (t-SNE) are examples of reduction techniques that were built to preserve clustering structure while performing dimensionality reduction [4, 19, 21]. These methods often do well at preserving local structure but t-SNE, for instance, can result in clusters where there were none before when a user neglects to properly tune the parameters for the process; this breaks the global structure.

Many look to UMAP to get around this issue, which it sometimes does, though there are claims that this can also falter under certain circumstances [13]. Nevertheless, this thesis attempts to improve upon such techniques by bettering the preservation of both local and global structures through the reduction process.

Dimension reduction methods utilize dissimilarity matrices that relate each node to the other points in the said point cloud. This dissimilarity matrix is often built by calculating the Euclidean distance between points. The reduction method then uses this dissimilarity matrix during its process to find the best relationships and build the dimension reduced result.

2.2.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is one of the dimension reduction algorithms that can help us better explore hyper-dimensional data. The idea behind this algorithm is to reduce the dimensionality of the hyper-dimensional data while preserving as much statistical information (variability) as possible [10]. To achieve its goal, the process of PCA works by recognizing orthogonal vectors in hyper-dimensional space that maximize the variance of the data. Since PCA is a linear algorithm, this can lead to quality issues with visualizations when dealing with manifold structures that are non-linear.

2.2.2 Multidimensional Scaling (MDS)

Multidimensional Scaling (MDS) is another dimension reduction technique, which aims to represent the dissimilarities between hyper-dimensional instances as distances between nodes in a lower-dimensional space, so the dissimilarities closely match the distances. This is accomplished by constructing a hyper-dimensional matrix to represent these dissimilarities of the data, which it then tries to maintain these dissimilarities as it scales down into a lower-dimensional space.

This method can be metric or non-metric [4]. MDS is more of a blanket title for similar but varying forms of the same method which have some slight semantic differences, however, the idea of scaling the differences down in dimensionality is generally the same. Classical MDS is almost identical to PCA [29], so the latter has been dropped in this paper for MDS. This similarity can be attested to in Figure 2.2.

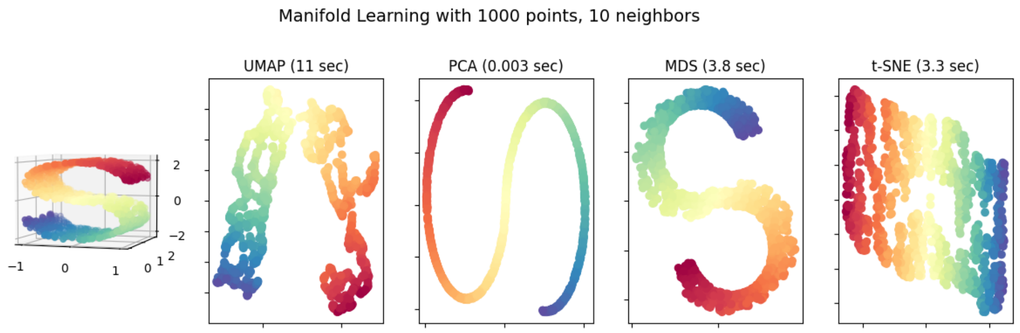


Figure 2.2: Reduction methods S curve. Here is an example of visualizations, produced after a sample S curve had been generated, of the dimension reduction methods mentioned in this namely UMAP, PCA, MDS, and t-SNE. These can be seen in this respective order, left to right, in the figure above. As mentioned, PCA and MDS produce similar results from dimension reduction, while t-SNE and UMAP have their own unique results. These two-dimensional reductions have been performed on the three-dimensional S curve seen on the left on the figure, which was generated using 1,000 points and using 10 neighbors for each point.

2.2.3 t-distributed Stochastic Neighbor Embedding (t-SNE)

Unlike MDS and PCA, t-SNE is a more recently developed non-linear algorithm also used to reduce hyper-dimensional data. Being a non-linear algorithm, t-SNE maintains the local structure of the data better than MDS or PCA, especially in manifolds that have a non-linear structure. This algorithm is often used to better maintain the local and global structure, when compared to non-linear techniques, though it excels at the former, not so much the latter.

Generally, t-SNE attempts to build a map between the low and hyper-dimensional distributions, which then produces a low dimensional set of data that closest matches the dissimilarities in the hyperspace data. t-SNE first calculates a distance measure between pairs of nodes in the original hyper-dimensional space data, as well as the low dimensional space data, which are then used to create a probability distribution that represents a node being another's neighbor. These calculated pairwise distances are what we refer to as the dissimilarities between the nodes. These high and low dimensional dissimilarities are then used to build a Gaussian and Student t-distribution respectively, which then uses a cost function to lead the algorithm into finding a map that best matches these distributions [20].

2.2.4 Uniform Manifold Approximation and Projection (UMAP)

Uniform Manifold Approximation and Projection (UMAP) has been said to be an algorithm to replace t-SNE, as it excels at memory optimization, embedding into higher dimensions for other uses than visualization purposes, and better preserving the global data structure [21]. One key difference between the two is that, unlike t-SNE which uses normalized dissimilarities from Euclidean distances, UMAP uses a non-normalized probability distribution where any distance can be plugged in. UMAP leverages this to be adaptive and maintain local connectivity of the manifold, giving a distance metric that varies from point to point. Essentially, UMAP ends up retaining more variation in the components of data, which results in better clustering. UMAP is often claimed to also maintain a better global structure, but there have been disputes to these claims [13].

2.2.5 Summary

As we have seen here, many algorithms attempt to preserve either local or global dissimilarities and structure. They also often do it by maintaining dissimilarities between pairs of instances in the higher dimensional space. In contrast to these methods, our method attempts to improve these approaches by better preserving both local and global structures through utilizing the dissimilarities between three or more nodes in the hyperspace. It is subjective to say which dimension reduction technique is inherently 'better' since not many

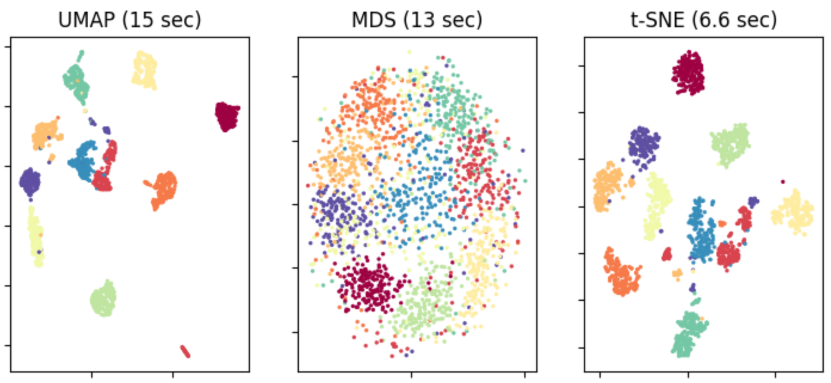


Figure 2.3: Elected reduction methods MNIST excerpt. Here is an example of visualizations of an excerpt from the MNIST dataset, procured by the dimension reduction methods we will be focusing on in this thesis: UMAP (left), MDS (middle), and t-SNE (right). UMAP and t-SNE perform well in regards to clustering the points, which can be attributed to their tendency to favor local structure in a hyper-dimensional point cloud reduction. MDS may look different from UMAP and t-SNE, however, different dimension reduction techniques prioritize certain structures and information than others.

ways exist to calculate a metric of visualization. In Figure 2.3, we can see that these dimension reduction techniques differ in structure and results. It is important to remember that while one intention of this thesis is to better preserve global and local structure throughout the performing dimension reduction, the visualization aspect can be subjective to the user.

2.3 Dimension Reduction Use in Visualization

Dimension reduction is important for data visualization for a vital reason. Reducing hyper-dimensional data lets us plot said data into a visualizable two or three-dimensional space. This allows one to be able to notice patterns in clusters and more, which gives us valuable insights into the data and its structure. Dimension reduction has been used for various visualization topics; Wang and Gu having used dimension reduction to visualize hyper-dimensional, noisy Single-cell RNA-sequencing data [28], and Joswiak et al. have used dimension reduction for hyper-dimensional data involved with chemical processing to help gather insights and troubleshoot the chemical process performance with visualizations [11], are some examples of the practice. There are many others who use dimension reduction for general visualization of complex hyper-dimensional data [3, 6, 22].

Using dimension reduction for data visualization is very useful, but it can also come with some drawbacks. Most notably, although we can use dimension reduction for data visualization, this process has the side effect of losing structure and other information of the original data. We should also take into consideration that dimension reduction can also falsely show structures that are not present in the data. This is a prevalent issue in dimension reduction in general, which our paper aims to better the approach to.e

Chapter 3: Improving Dimension Reduction Using Hyper-Dimensional Spanning Trees

Now that we have explained what the issues of dimension reduction in visualization are, as well as given some background information on the subject, we can talk about the steps of our algorithm in more detail. The process we have created attempts to improve current dimension reduction techniques by modifying the underlying data each algorithm uses in order to emphasize important structures in the data.

3.1 Gathering k-Nearest Neighbor Data

The first step of the process is to obtain the k-Nearest Neighbors (KNN) graph from our hyper-dimensional data. The imported original data is converted into hyper-dimensional points inside a hyper-dimensional point cloud, where the values of the original node's vectors act as the hyper-coordinates in this point cloud; these hyper-dimensional points are then added to a new graph which will represent the original dataset. We utilize a KNN graph for the later construction of our MST. A KNN graph is built by connecting each node in a graph to its respective k 'nearest neighbor' node(s). The nearest neighbors of a node are found by calculating the distance from the node to each of the other nodes.

In this implementation, we convert hyper-dimensional data into nodes of a graph, using the hyper-coordinates calculated from the data's values. We then use a KDTree, and the user-specified amount of neighbors k, to calculate the distances between the hyper-dimensional node neighbors of each hyper-dimensional node in the point cloud. Once these distances are calculated, edges are added to the graph which connects the node to the k

nodes that are closest to it. This is repeated for each node of the graph. An illustration of this KNN graph building process can be seen in Figure 3.1.

A pitfall to avoid when building a KNN graph, is to avoid high space and time complexities when searching for each instance's respective neighboring instance(s). We choose to use a KDTree of our data, rather than brute-forcing the KNN graph, as it has been proven to be an efficient way to query for information regarding the nearest neighbors of any given instance [1], allowing us to build a KNN graph in a fast manner. Depending upon the data and the number of selected neighbors, k, this could result in a KNN graph that has just one component, or a graph which many components, connecting each of the nodes to at least k other nodes. Ideally, k should be selected that it is not too small to result in a sparse graph, or a k that is too large and results in a dense graph, slowing down the program unnecessarily. Selecting an ideal k can be challenging and may require utilizing the user's prior knowledge about the data, which may take some experimentation. The edges in this graph are the smallest edges the original point cloud potentially had.

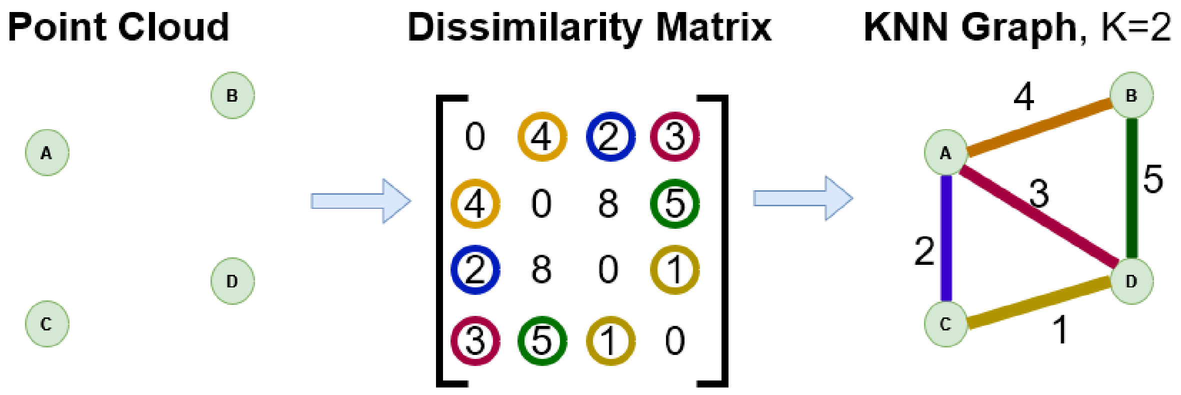


Figure 3.1: Building a KNN graph. Illustration of the conversion from a point cloud to a k-Nearest Neighbors graph. An arbitrary point cloud (left) is used to calculate a dissimilarity matrix (middle) to represent a metric, such as the pairwise Euclidean distances between the points. This is then used to build a graph that connects each node to its 2 nearest neighbors, resulting in a k-Nearest Neighbors graph (right).

3.2 Building a Hyper-Dimensional Spanning Tree

From here, we begin to build a Minimum Spanning Tree (MST). This can be a normal spanning tree in the same dimensionality as nodes and edges are, but it can also be a hyper-dimensional spanning tree. A conventional MST is formed by a subset of edges from a connected, weighted, and undirected graph which connects all of the vertices in the graph. The MST can be constructed using a few differing algorithms, a couple common algorithms being Prim's algorithm [24] and Kurskal's algorithm [16]. Arguments can be made for both, especially regarding performance on a dense graph, but their ideas are similar: add the smallest possible edges from the original graph, until all nodes are accounted for, while avoiding cycles. The time it typically takes to generate an MST of a graph is based on its number of edges and nodes. For example, an MST can be built with Kruskal's algorithm at $O(E \cdot log(V))$ time, where V and E are the number of vertices and edges in the original graph respectively. A conventional MST can be used for our algorithm, but let us focus on how the hyper-dimensional spanning tree is constructed.

A hyper-MST can be thought of as an MST that instead of a spanning tree that is built from connections between two nodes (edges), is built from connections that relate three nodes or more to each other. For our conventional MST, we use the KNN graph to build the MST using two-point connecting edges. For our hyper-MST, we want to utilize the 3-cliques of the KNN graph, which consists of six edges, to reinforce the relationships between three nodes. This relational information would give the algorithm more information on the structure of the graph than if we only used an MST which was built on the principle of looking for relations between two nodes.

Usually an MST can be constructed by selecting the smallest edges in a connected KNN-graph until all nodes are connected and accounted for. A hyper-MST can be constructed by discovering specifically sized cliques in the original KNN graph. These cliques are what will act as the hyper-edges in the hyper-MST. Instead of representing the relations between two instances or nodes, as a typical edge would, these hyper-edges give us information about the relations between several nodes. This 'hyper-MST-2' represents 3-cliques of nodes in the original KNN graph as hyper-edges or triangles, and where the original edges are represented as hyper-nodes. This could also be taken to a further degree, to represent connected triangles as tetrahedrons in 4-cliques as a hyper-MST-3, and so on. This can be abstracted to any n-cliques, which will be used to construct some hyper-MST-m, where $m = n - 1$.

All edges from the KNN graph are then imported to act as the hyper-nodes of the next order MST. We then attempt to generate all possible hyper-edges between these hyper-nodes, but only keep the ones that are solely KNN-graph edges that make up a clique, i.e., not a set of hyper-nodes whose corresponding base edges are non-cyclic. The weight of these hyper-edges becomes the greatest value of the edges that make them up.

The construction of the hyper-MST utilizes 'hyper-edges' in order connect all hyper-nodes and components. Hyper-edges, in the context of this thesis, are an abstraction of M-vertex cliques in the initial KNN graph of the higher dimensional data, where M is the number of vertices that can be modified to only count cliques of certain sizes in the higher dimensional KNN graph. In our approach, we transform a graph of edges into a hypergraph where the base edges are represented by hyper-nodes, and where the hyper-edges are a representation of the cliques in the original graphs. These hyper-edges will store the clique as a set of the original edges, now hyper-nodes. The hyper-nodes will store the original nodes that make up these original edges, to be used for modifying the dissimilarity matrix with its relationship data. It is also important to note the weight of the hyper-edges will be calculated as the highest weight of the edges that comprise them. An illustration of this process can be seen in Figure 3.2.

The hyper-edges allow us to deduce more information about larger amounts of nodes, rather than solely pairs. In a sense, this method can tell us which nodes are more similar to each other, but perhaps more importantly, can concurrently tell us about the dissimilarity

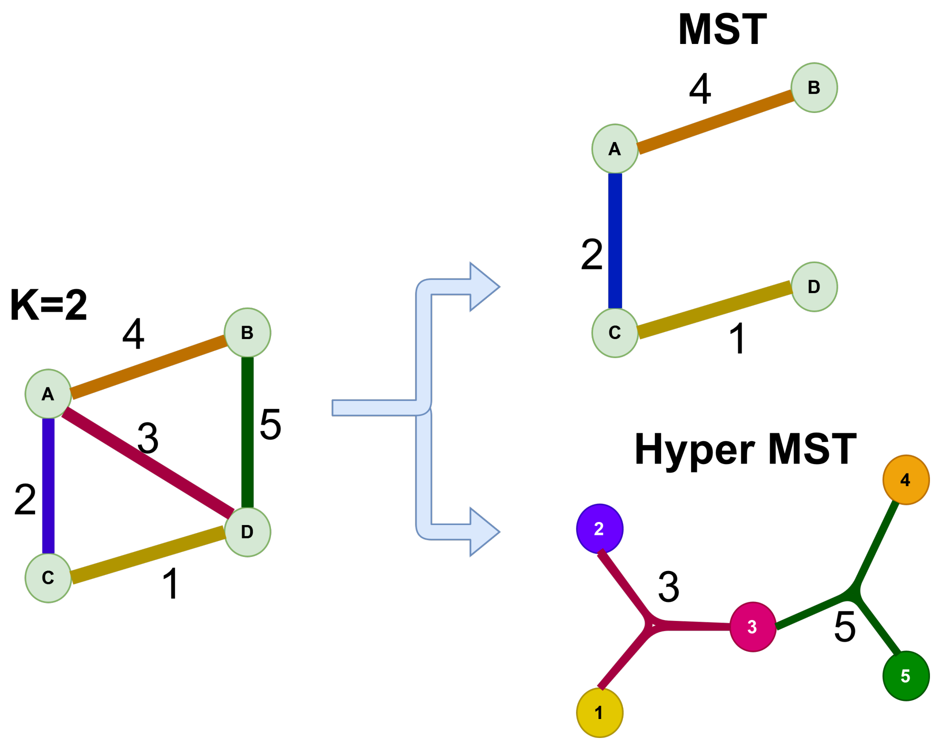


Figure 3.2: Building MSTs using KNN graphs. Illustration of the conversion from a KNN graph to an MST. The KNN graph can be used to build the MST of some arbitrary point cloud (top), given that the number of neighbors used is sufficient enough to build a connected graph. The KNN graph can also be used to deduce the M-vertex cliques in a point cloud (bottom), thus allowing a hyper-MST graph to be built via hyper-edges, representing more information of the most pertinent relations of the original point cloud.

between nodes as well. The idea behind this is that this information will preserve more information and consequently, better preserve the structure of the original data.

To reiterate, the benefit here is that we can use these hyper-edge cliques to retain more information of the initial hyper-dimensional data graph, thus retaining more structure. The drawback here is that while these hyper-edges can indeed give us more information about the relations between instances, there is quite an increase in processing time.

For an example, let us look at an MST that is constructed using cliques of size 3. We have our initial KNN graph with some edges depending on the parameter which defines the number of neighbors. We then add all the KNN graph edges as hyper-nodes to a new graph, the hypergraph MST. We then generate all valid hyper-edge/triangles or 3-cliques between these nodes. Our current implementation approaches this by attempting to generating all possible hyper-edges/triangles between all hyper-nodes/nodes, but keeping solely the valid triangles, then building the MST from these triangle hyper-edges. The drawback of this implementation means that for a base graph with N-1 edges, we end up with essentially N^2 hyper-edges, before checking each one's validity; this extends the processing time. After all valid hyper-edges are found, we then use them to build the hyper-MST.

This example hyper-MST is just the first step. This was generated from finding cliques inside the base level KNN graph. However, we can compound on this by finding larger cliques, such as 4-cliques (tetrahedrons), which can give us more information for the instances of the graph. However, it is also important to realize that at some point there will be a diminishing return, especially when compared to the processing time and power of such a program. With a high enough k value used to generate the KNN graph, a higher clique value could also end up relating nodes that did not have a strong relation before, as seen in Figure 3.3.

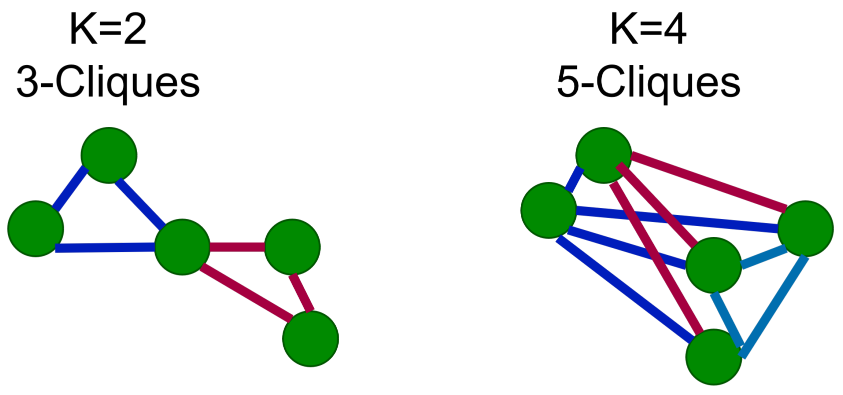


Figure 3.3: Hyper-MSTs with different clique sizes. This figure shows how choosing a higher clique value could actually harm the construction of a hyper-MST. The figure where k=2 (left) gives us two hyper-edges and relates two groups of KNN nodes. The figure where k=4 (right) shows a hyper-MST with one clique, and one resulting hyper-edge. The weight of a hyper-edge is defined as the highest weight edge that comprises it. This means that choosing a higher clique value is not always beneficial, as it causes all relationships within it to reflect the higher weight and weaker relationship.

As seen, there are a few ways we can modify the MST that is output from this stage in the program. However, whichever order of the spanning tree is constructed, each one is going to be used to influence the dimensionality reduction algorithms.

3.3 Selecting Important Features From the Hyper-MST

The user also has an option to apply some thresholding to select which components are used within this hyper-MST. This option is presented as a barcode and allows the user to select a range of the most important components from the hyper-MST to apply, if they choose to not apply all of the components in the hyper-MST to the algorithm. The barcode represents features within a graph based on their significance [26]. We can think of graphs as an abstract representation of the relationship between the instances in the graph. After reduction and visualization, we should end up with a graph that effectively presents structure to give insights of the data. We then extract these features and produce a barcode which allows for the user to manipulate which features of the data to use for the influencing of the dissimilarity matrix.

3.4 Influencing Dimension Reduction Algorithms with MST

Once the hyper-MST is created, to the specified dimensionality, we move to the next step of the process. We then take the newly generated hyper-MST and use it to procure our own dissimilarity matrix for the dimension reduction algorithms. As mentioned prior, these methods operate in a manner that prioritizes pairwise differences between instances. Given a hyper-MST, we modify the usual dissimilarity matrix into one that has information about 3-cliques of nodes, rather than solely pairs. This helps the algorithm to emphasize the similarities and dissimilarities throughout the original data, as we can modify the length to be smaller or greater than initially calculated, using this extra information.

3.4.1 Modifying the Dissimilarity Matrix

We then generate the dissimilarity matrix using the default Euclidean values, as it would if it were to progress normally. Now we start to modify the dissimilarity matrix using the MST data we also have. For the initial n^2 matrix we have, we choose to change the connections that we have in this MST. For each hyper-edge of the MST, we modify the dissimilarity values of the KNN-graph to be smaller or larger, by referencing which nodes and relationships in the KNN graph we want to modify; this process can be seen in Figure 3.4. We can also see in Figure 3.4 that sometimes hyper-edges overlap in which KNN graph edges they are comprised of. This could lead to the same relationships being modified multiple times. However, we account for this by solely allowing the lowest-weight hyper-edge to affect the relationship.

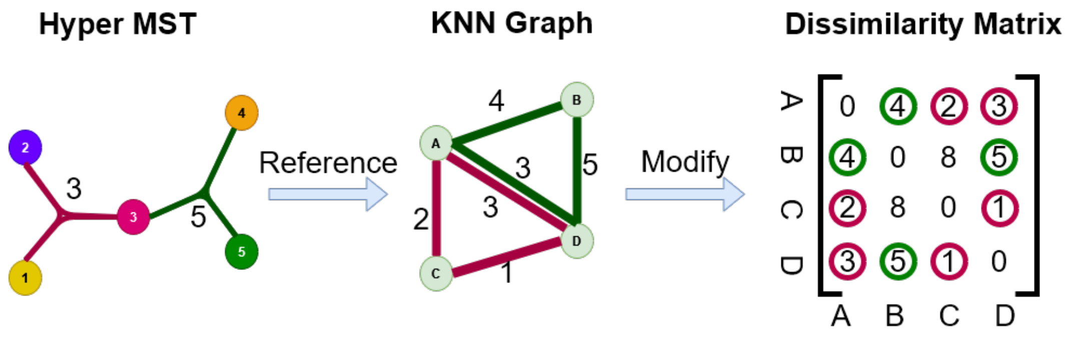


Figure 3.4: Using hyper-MST to influence the dissimilarity matrix. To modify the original dissimilarity matrix (right), we have to utilize the hyper-MST. The hyper-edges in a hyper-MST (left) allows us to reference the original KNN graph (middle); this tells us which relationships of the original nodes to modify. We then index these nodes in the dissimilarity matrix and modify their values, which will influence the dimension reduction methods when passed in.

In reference to hyper-dimensional data, we want the edges of the MST to change the dissimilarity matrix of the hyper-dimensional data. We set a range in this algorithm, [a.b], and we transform the dissimilarities in this matrix based on a linear mapping ($T : R^n \to R^m$ where $T(ax) = a \cdot T(x)$) of the range of the edges in the MST to this set range. For example, if we have edges or hyper-edges in the MST that range in value from 100 to 200, with a linear mapping to a range of say [0.5, 1.5], the edges that are valued at 100 will shrink the matching values in the dissimilarity matrix by multiplying it by 0.5, but a value of 200 in the MST means the values in the dissimilarity matrix will be enlarged by multiplying the corresponding values by 1.5. In other words, we try to enhance the structure of instances that are near and far from each other by transforming the dissimilarity matrix in this manner.

3.4.2 Effect on the Reduction Methods

After this dissimilarity matrix is complete, the dimension reduction algorithms carry on as normal, though using our custom modified matrix. This approach should result in a dimension-reduced set of data that has been improved from conventional means when displayed visually.

When we have the modified dissimilarity matrix through the use of our MST, hyper or regular, we plug this dissimilarity matrix into the reduction algorithm with its 'metric' parameter set to our precomputed matrix. At this point, the only difference between the default reduction algorithm and ours is that we have a modified version of the dissimilarity matrix for the algorithm to use. Since our precomputed dissimilarity matrix has 'enhanced' the most important near and far information between instances, the algorithm should take this information and produce a dimension-reduced result that has better preserved the local and global structure of the original hyper-dimensional data.

3.5 Preservation of Structure

When the algorithm is finished, we want to be left with a visualized graph that effectively communicates the structure of the data. We believe that by emphasizing the more important features of the original data through the hyper-MST influencing the dimension reduction methods, we will be left with such a more effective graph visualization than without our changes. A strength of our method is that it does not modify nor mangle the underlying structure of the data. Our algorithm simply enhances the most prominent information and features through the use of our MSTs. It is also important to realize that we do not intend to modify too many dissimilarities of the matrix, nor do we modify them in a way that escapes the range of the original data, rather we emphasize a range of the most important features slightly more than the less important features.

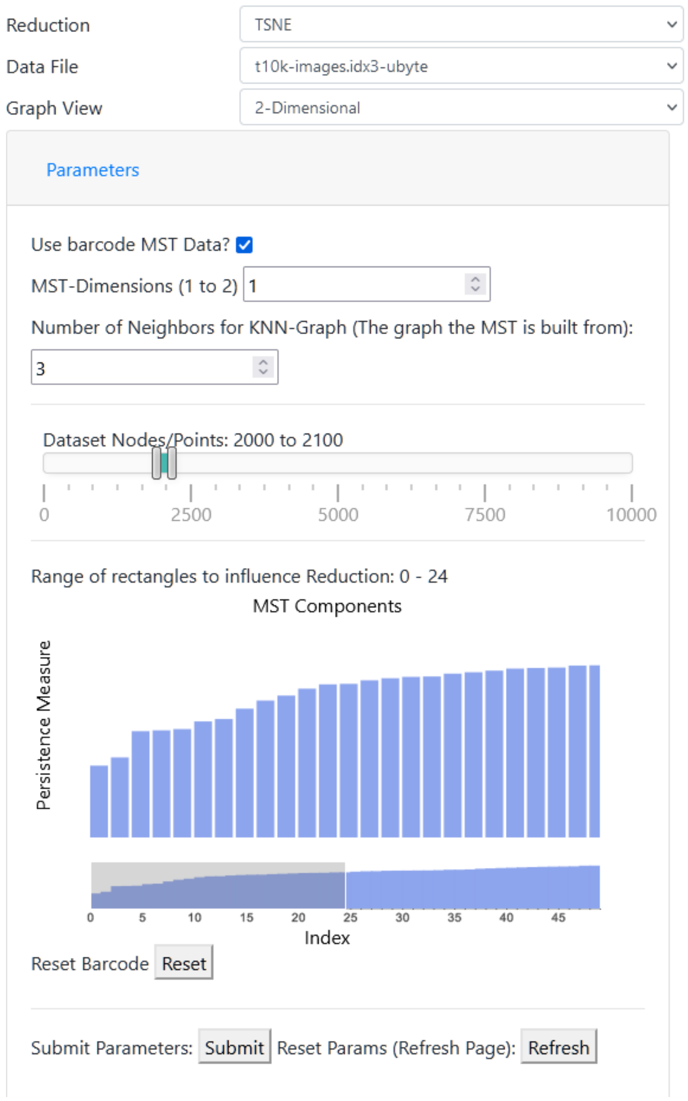
Chapter 4: An Implementation of the Algorithm

Our implementation of the algorithm is set up to operate using a web interface. The program runs on a local server, which uses the Python flask framework, where the webpage is presented using JavaScript (JS), HTML, and some CSS styling. This webpage interface is what allows the user to set their parameters to the algorithm, as well as view the resulting dimension reduced graph and the hyper-MST. The backend of our implementation, where we process these parameters and where our algorithm operates, is written in Python. After the program goes through its process and the dimensionality is reduced and MST graphs are finished, they are passed back to the front end. We use the implementations of the MDS and t-SNE dimension reduction algorithms in the `scikit − learn` [23] Python library. We also use the `umap − learn` [14] implementation of the UMAP algorithm. We then use the Data-Driven Documents (D3.js) JS library [2] to build our visualizations of the dimension reduced and MST graphs.

4.1 The Web Program

A screenshot of the UI can be seen in Figure 4.1. This webpage presents the user with a few choices for parameters:

- Reduction Method, lets users choose between t-SNE (the default), MDS, or UMAP;
- Data File, lets users choose the dataset to run the algorithm on;
- Checkbox to toggle between regular dimension reduction and our approach;
- Number of Neighbors for KNN-Graph, which is used for the construction of the hyper-MSTs. The number of neighbors chosen can affect the resulting amount of hyper-edges


Reduction: TSNE

Data File: t10k-images.idx3-ubyte

Graph View: 2-Dimensional

Parameters

Use barcode MST Data? ✓

MST-Dimensions (1 to 2): 1

Number of Neighbors for KNN-Graph (The graph the MST is built from): 3

Dataset Nodes/Points: 2000 to 2100

Range of rectangles to influence Reduction: 0 - 24

Figure 4.1: Webpage interface parameters. A screenshot of the parameters the user is allowed to modify. Most of these parameters pertain to values or options that affect the hyper-MST directly, while there are also parameters to specify which algorithm to use, which part of the data to extract, and one that toggles our algorithm's functionality.

and information the hyper-MST supplies to the dimension reduction process. The more neighbors here, the better the resulting MST is, with the drawback of increasing the processing time, potentially dramatically;

- Range of Dataset Instances, allows user to select the subset of data to extract and operate on from the dataset specified. This is a capped range, as larger values may take increase the time for the algorithm to run and negatively impact interactivity;

- Barcode slider, to specify some threshold of choosing the more important hyper-edges for the process. This parameter intends to aid in the modification of the dissimilarity matrix, by specifying the more important components that we wish to act upon in the algorithm. For example, if we wish to worry about the most prominent and important cliques of the KNN-graph, we can choose to modify only the most important dissimilarities of our computed matrix, to study the effects of using some information from the hyper-MST, rather than all of it;
- Buttons to submit parameters. After the user submits their choices, the process begins with accepting the parameters the user passes in. These parameters are then sent to the backend of the program for processing.

4.2 Importing Data

As discussed, we primarily have tested our method on the MNIST and Iris datasets, though there are others available to choose from. The MNIST dataset is comprised of 28-by-28-pixel images, which are transformed into 784-length vectors, where each value represents one pixel. The Iris dataset is a dataset comprised of instances in a 5-length vector form. Each of these vectors is used to represent hyper-coordinates in the hyper-dimensional space. The raw data from these files are parsed and loaded into hyper-dimensional point clouds.

4.3 Building the KNN Graph

Once we have obtained the instances and their classes, we can move on to generating the KNN graph of this data. We do this by using the KDTree method of sklearn, with a Euclidean metric. This method allows for a KDTree to be built using hyper-dimensional points or instances and allows us to efficiently extract the neighbors of any given instance.

Once we have the ability to query the KDTree for neighbors of any instance, we can build the KNN graph. This process is also dependent on a parameter, namely the k chosen by the user. If too high of a parameter is given here, it could cause the processing time of the program to slow drastically. k can also affect the quality of the graph; a k that is too small, and we will have a sparse and potentially ineffective hyper-MST with many components to use with our algorithm. There is a middle ground for k, that may need to be experimented with depending on the data used, to produce a quality hyper-MST in acceptable time.

This is simply done by creating instances in a NetworkX graph, which holds the instances as nodes. We then iterate through the nodes and add edges to the K-Nearest Neighbors of each node. This gives us the final KNN graph.

4.4 Building a Conventional MST

In this program, the user must decide whether to use an MST to influence the process and if so, to what level of an MST we build.

Let us discuss the scheme of building a conventional MST (MST-1). The idea of this MST is simple: use the KNN graph to build an MST for the hyper-dimensional data. The function to build MST-1 utilizes the KNN graph, the range of nodes, and the thresholding range if specified by the user. Here, we use NetworkX's tree functionality and input the KNN graph into the `minimum_spanning_edges` function to extract the MST. Once we have these edges, we can sort them if needed, then apply the thresholding on them as specified by the user-chosen parameter. This threshold range comes directly from the barcode slider to select the more important edges of this MST.

One thing to note here is that a completely connected MST may not be possible depending on the k chosen by the user in this part of the process. For instance, if the user chooses a k that is too low, we may end up with a KNN graph that is not fully connected but with multiple components. If the KNN graph passed in is connected, the `minimum_spanning_edges` function finds the appropriate MST. If it is not connected, the function finds a spanning forest instead. Nevertheless, the components will still be applied to the dimension reduction process.

4.5 Building the Hyper-MST

The other MST we can use for dimension reduction influencing is a hyper-MST, specifically the hyper-MST-2 which is the next dimensionality MST we can create. A hyper-MST is built by extracting cliques, instead of edges in a normal MST, from the KNN graph. In this hyper-MST-2, we choose to extract 3-cliques, which are analogous to triangles. The 3-cliques are found in the KNN graph by using a NetworkX function that finds cliques, `enumerate_all_cliques`, which we specify to find and then store cliques of size 3. We store these cliques with the respective KNN nodes and edges that comprise the said cliques, as well as the max edge weight of the clique. The max edge weight will act as the weight of the hyper-edge in the hyper-MST.

Once we have these cliques extracted from the KNN graph, they will be inserted into the hyper-MST graph as hyper-edges. The hyper-MST is implemented as a NetworkX graph in Python. To add hyper-edges to this data structure, we simply add extra data to a normal source-and-target edge of the NetworkX graph, specifically we add a third node for a 3-clique along with the standard source node, target node, and the hyper-edge weight. One thing to also note is that after this process of adding all of the possible cliques as hyper-edges, we now have many hyper-edges, some of which that share KNN nodes with other hyper-edges; this can cause many excess hyper-edges to exist, meaning that they will be removed when constructing the minimum spanning tree.

The hyper-nodes here are the KNN graph's edges; for this implementation, these are connected 3 at a time in the hyper-MST-2 since we specified to use cliques of size 3. There are two separate paths we can take from here. For one, we can build the hyper-MST to account for most of, or possibly all, KNN nodes which currently make up all of the current hyper-edges, allowing us to remove the so-called 'duplicate' hyper-edges. We could also build the hyper-MST to satisfy all of the hyper-nodes in the hyper-MST. For this implementation, we chose to build the MST to satisfy as many hyper-nodes as possible. Keep in mind, that the resulting hyper-MST may not connect all KNN nodes.

A hyper-MST could hypothetically continue to increase in dimensionality, but some problems arise from this. For one, as we increase in the dimensionality of hyper-edges from triangles to tetrahedrons and so on, we are technically saving components that give us information about an increasing amount of original hyper-dimensional instances. We want to find a middle ground hyper-MST that gives us thorough information about the structure of the hyper-dimensional data, without resulting in a hyper-MST that has few edges connecting many instances. Obtaining such a hyper-MST would compromise the structural data we are trying to preserve. A graph's structure is not derived from a few solid constants, but rather a slew of dynamic and variable components that act together to represent the skeleton of the relations in the data.

4.6 Influencing the Dissimilarity Matrix

The chosen MST and KNN graphs are then passed into the reduction function of the process. The other parameters are passed in to specify the number of dimensions we want to reduce to and the reduction method to use. At this point, we want to have the original point cloud, including the classes and coordinates of the respective instances. This will be used to build a dissimilarity matrix, as well as feed the data into the reduction methods.

After we have the initial Euclidean dissimilarity matrix, we can use the hyper-MST we have built to influence the reduction method by modifying this dissimilarity matrix. We use either the MST or hyper-MST for this part of the process, but we will focus on the hyper-MST, as both are utilized similarly.

We iterate through the hyper-edges of the hyper-MST and take note of the KNN nodes that comprise each hyper-edge. To make this a more arbitrary step, we take note of these by storing this as a component, which stores only the KNN nodes of the hyper-edge, along with the hyper-edge weight. Converting edges and hyper-edges into these components allows for any MST to be primed for this part of the process.

We now have a set of components that are marked to be modified in the Euclidean dissimilarity matrix. Iterating through the components, we extract the KNN nodes that comprise each component. We then map this to the respective entries in the dissimilarity matrix. For example, a 3-clique hyper-MST would result in three entries of the matrix to be modified for its three edges, a 4-clique would modify 6 entries, and so on.

To modify these entries, we set a linear mapping to transform the existing values in the dissimilarity matrix, based on the minimum and maximum values of the hyper-edges in the hyper-MST. We then multiply the dissimilarity matrix values by a number in the range [0.5, 1.5]. The idea here is that larger edges in the hyper-MST will emphasize the larger connection by making the respective dissimilarity matrix values larger, and emphasize closer instances by shrinking values in the dissimilarity matrix that correspond to the lower values in the hyper-MST. This range could affect how the hyper-edges emphasize similarity in our resulting graph, but this implementation will be based on and interpreted with this range. In essence, we are transforming certain dissimilarities in the matrix by using the hyper-edges of the hyper-MST, using a linear mapping to either shrink or enlarge the precomputed values.

4.7 Dimension Reduction

After we have the dissimilarity matrix modified by our MST, we take it and plug it into the dimension reduction techniques we have imported from various Python libraries. t-SNE and MDS are provided by the `scikit − learn` library, while UMAP is from the `umap − learn` library. Most of the parameters of these functions are the default values, though some have been changed. For one, for each of these algorithms, we supply them with the number of components, which is the dimensions of the data the user elects to reduce to. For UMAP, we elect to specify the `min_dist` parameter to `0.3`, in order to provide clearer visuals; this parameter modifies the minimum distances between points in the resulting graph, where lower values would tightly pack the points together. Other than these specifications, we either allow the dimension reduction functions to run using their default Euclidean matrix, or we supply our modified one.

After these dimension reduction methods are complete, we are then left with the embedded data. We then evaluate these results using our chosen performance metrics and store the data. The reduced data is then returned to the server, where it will be passed back to the front-end of the web application to be processed for visualization.

4.8 MST Component Barcode Slider

The process of creating the MST also results in producing a Barcode Slider, which is presented to the user, after the process is finished. The most prominent features, the hyper-edges, are extracted from the hyper-MST we have generated, and visualized in a barcode format. This barcode range-slider intends to let the user select which of the edges or hyper-edges to use to modify our dissimilarity matrix. For instance, maybe the user does not want to use all said edges or hyper-edges, because they believe the more important closer edges should be used and the more distant ones are less important to the process; this slider gives the user that freedom. An example of the barcode slider can be seen in Figure 4.2.

4.8.1 Thresholding with the Barcode Slider

The idea behind using the barcode is to represent the most important features in the MST for our data and allow the user to select which to use during the part of the process where the MST influences the dissimilarity matrix. More specifically, the persistence barcode represents the hyper-edges inside our hyper-MST and their significance to the data. The barcode we have implemented allows the user to interactively select a range of the more important hyper-edges, and submit this to the program as an additional parameter. Once the algorithm reaches the point of modifying the dissimilarity matrix, this range parameter acts as a threshold to which hyper-edges influence the dissimilarity matrix. For instance, the user could choose to use solely the first 25 hyper-edges of the hyper-MST to apply to our

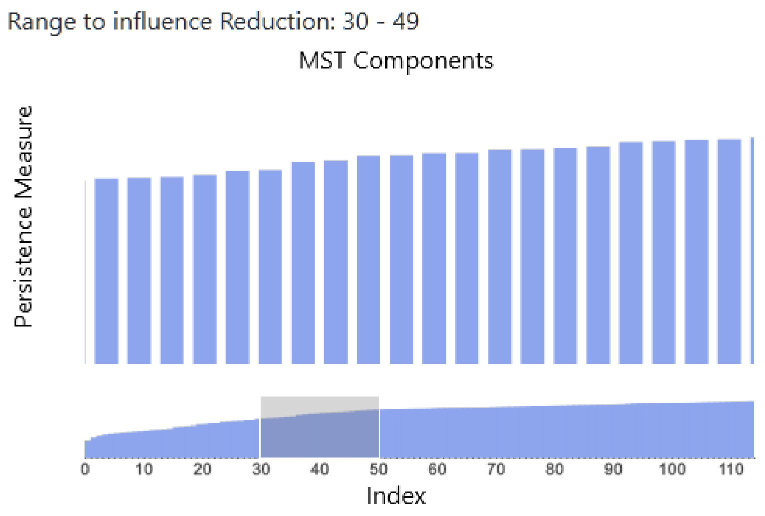


Figure 4.2: Barcode slider. A screenshot of a barcode after running the algorithm with some arbitrary parameters, with our algorithm enabled. The resulting barcode allows the user to select which hyper-edges and components to use during the dissimilarity matrix modification by dragging the grayed-out slider below the barcode.

dissimilarity matrix, which will result in a graph that is different than when all hyper-edges are utilized. We hope that by using this barcode slider, the user can have more freedom and better results when attempting to study the significance of the features of their data.

4.9 Visualizing the Data

The server passes the dimension reduced data, as well as the hyper-MST data, to two separate functions which we have implemented using the `d3.js` JavaScript library.

Firstly, the dimension reduced data is plotted on a two-dimensional Cartesian coordinate plane. The dimension reduced data gives us the two-dimensional coordinates for each point, as well as their class. We use this information to plot the points appropriately, and assign it a color based on its class. This allows the user to see which points are clustered and separated in the reduced data. An example of this graph can be seen in Figure 4.3.

We also provide a visualization of the hyper-MST used for the dimension reduction when our algorithm is enabled. This hyper-MST is represented in a Force-Directed Layout

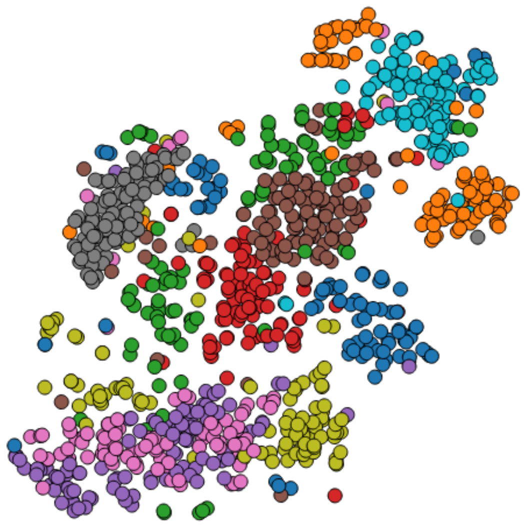


Figure 4.3: Dimension reduced data graph. An example screenshot of some dimension reduced data, plotted for the user to see.

(FDL) graph, which is an animated graph where the nodes and edges have a 'force' between them. This feature allows one to see a graph in an interactive manner while avoiding clumping edges and nodes together, which would be more difficult to visually decipher. The user is also able to interact with the hyper-MST graph and gather information from it. An example of the FDL graph can be seen in Figure 4.4.

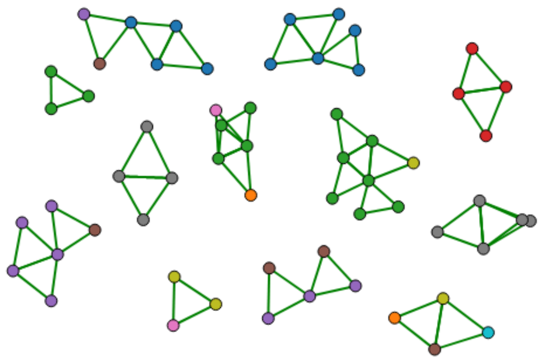


Figure 4.4: Force-directed layout graph. An excerpt from the Force-directed Layout Graph of the hyper-MST. This allows the user to see the hyper-edges and nodes that were constructed for the hyper-MST. For example, connections of 3-cliques can be seen in this image. This is also an example of when the hyper-MST is not fully connected

Chapter 5: Experiments

5.1 Datasets

For our experiments, we chose two datasets that would procure a visible structure, after being reduced, which can be seen to appropriately cluster if reduced properly. Namely:

- MNIST [17], which is a database of handwritten digits. The training set of this dataset is comprised of 60,000 examples, while the test set is comprised of 10,000. Each instance of the data is a black and white, normalized 784-dimensional pixel box of handwritten numbers from hundreds of people. Ideally, after the data is processed through dimension reduction, the digits should cluster near each other, after reducing from 784 dimensions. This clustering is a way to measure the success of retaining the original hyper-dimensional structure. An example of this dataset can be seen in Figure 5.1.

- Iris [8], is a well-known database in the field of pattern recognition. Each instance of the data is meant to represent an iris plant, which has five attributes in the data, including one 'class' attribute; there are 50 iris plant instances, per each of these class attributes. The associated task of this dataset is to be used for classification. This is a hyper-dimensional dataset, though it is a lot smaller compared to others such as MNIST, at five dimensions.

5.2 Performance Metrics

It is difficult to evaluate how well a dataset is visualized, especially when factoring in the dimension reduction aspect to these experiments. In light of this, we have selected a few

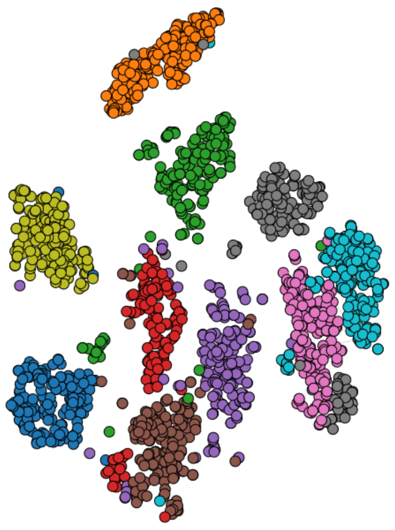


Figure 5.1: Example of MNIST dataset reduced through t-SNE. This figure illustrates a sample of the MNIST dataset, reduced using t-SNE, as well as using our implemented changes and algorithm. As seen, reducing the dimensionality of MNIST should gravitate towards producing ten separate clusters, where each node is an instance from the dataset of ten possible numbers.

specific metrics that we believe together will allow one to see what effect our implementation has on current reduction techniques, namely:

- Spearman rank correlation of the Shepard diagram, which is a measurement to track the correlation between pairwise distances in the hyper-dimensional and low-dimensional data. This score should increase as close to 1.0 as possible to be seen as an improvement for structural preservation [9, 27];
- Davies-Bouldin (DB) index, which is used for measuring how well clusters are formed from quantities and features inherent to the dataset [5]. This could be seen as a more visual gauge and a decrease should be seen as an improvement;
- Silhouette score, a measurement for clustering algorithms based on the mean intra-cluster distance and the mean nearest-cluster distance [25]. This measures cohesion and separation for the clusters, which could also be used to gauge visual results and an increase should be seen as an improvement;

- Q_{Local} and Q_{Global}, which are coranking measurements between the hyper-dimensional and low-dimensional data to measure local and global structure by averaging the number of overlaps between the k-nearest neighbors in hyper-dimensional and low-dimensional space [7, 15, 18]. These metrics are used to help gauge structural preservation and an increase should be seen as an improvement to the dimension reduction process.
- Runtime, the time in seconds it took for each experiment to execute and produce results.

We will also be keeping track of the parameters used for each experiment, to highlight differences for the outcomes and resulting performance metrics; this will also allow for one to reproduce our findings. Specifically, we will track the:

- Dimension reduction algorithm used;
- Dataset used;
- Range of nodes from the specified dataset;
- k for k-Nearest Neighbors;
- MST order, which can be None, set to MST-1, or set to Hyper-MST-2;
- Barcode thresholding range.

The implementations of MDS and UMAP have a random state initialization and produce varying results. To compensate for this, each experiment instance was run 10 times and averaged.

5.3 Baseline Experiments

In this section, we show each dimension reduction algorithm with either no MST applied, the MST-1 applied, or the hyper-MST-2 applied, as well as their respective resulting performance metrics. Figure 5.2 shows some baseline reference visuals of the tested dimension reduction algorithms, namely: t-SNE, MDS, and UMAP from top to bottom. Each of these

Table 5.1: Baseline t-SNE metric results. These are the resulting t-SNE metrics for the baseline experiment.

Baseline Results for t-SNE

Dataset	MNIST			Iris		
MST Applied	None	MST-1	Hyper-MST-2	None	MST-1	Hyper-MST-2
Spearman	0.5078	0.4956	0.4528	0.9708	0.9728	0.9749
Davies-Bouldin	1.3783	1.4889	2.0292	0.5009	0.5379	0.4923
Silhouette Score	0.3294	0.2978	0.2775	0.6372	0.6142	0.6431
Q_{Local}	0.5368	0.5405	0.5215	0.6509	0.6299	0.6276
Q_{Global}	0.4664	0.4744	0.4672	0.3978	0.3848	0.3807
Runtime (s)	7.2723	9.2752	10.7982	1.0033	0.9934	1.0838

rows shows an ordinary visual of reduced MNIST data, one with MST-1 applied, as well as our hyper-MST-2 applied. These visuals can act as a baseline reference to show what visual changes these influences impose on the current techniques.

These visuals can be used to gain an arbitrary sense of visual improvements from our algorithm. For instance, subfigure (i) of Figure 5.2 can be seen to remove splits between a class of nodes versus the subfigure (g) which has this issue. It is important to remember that this is a visual change, and though some changes may appear better or worse to the eye, the affect the algorithm has on the structure of the data could be the opposite. These visuals help us to understand the data, but should also be combined with our performance metrics when drawing conclusions.

The visuals are a good way to see changes between reduction algorithms and our algorithm. Table 5.1, Table 5.2, and Table 5.3 show the resulting performance measurements using our specified metrics. The results in these tables are also our baseline control experiments, where the only parameter changing between tests is whether a conventional or hyper-MST-2 is applied.

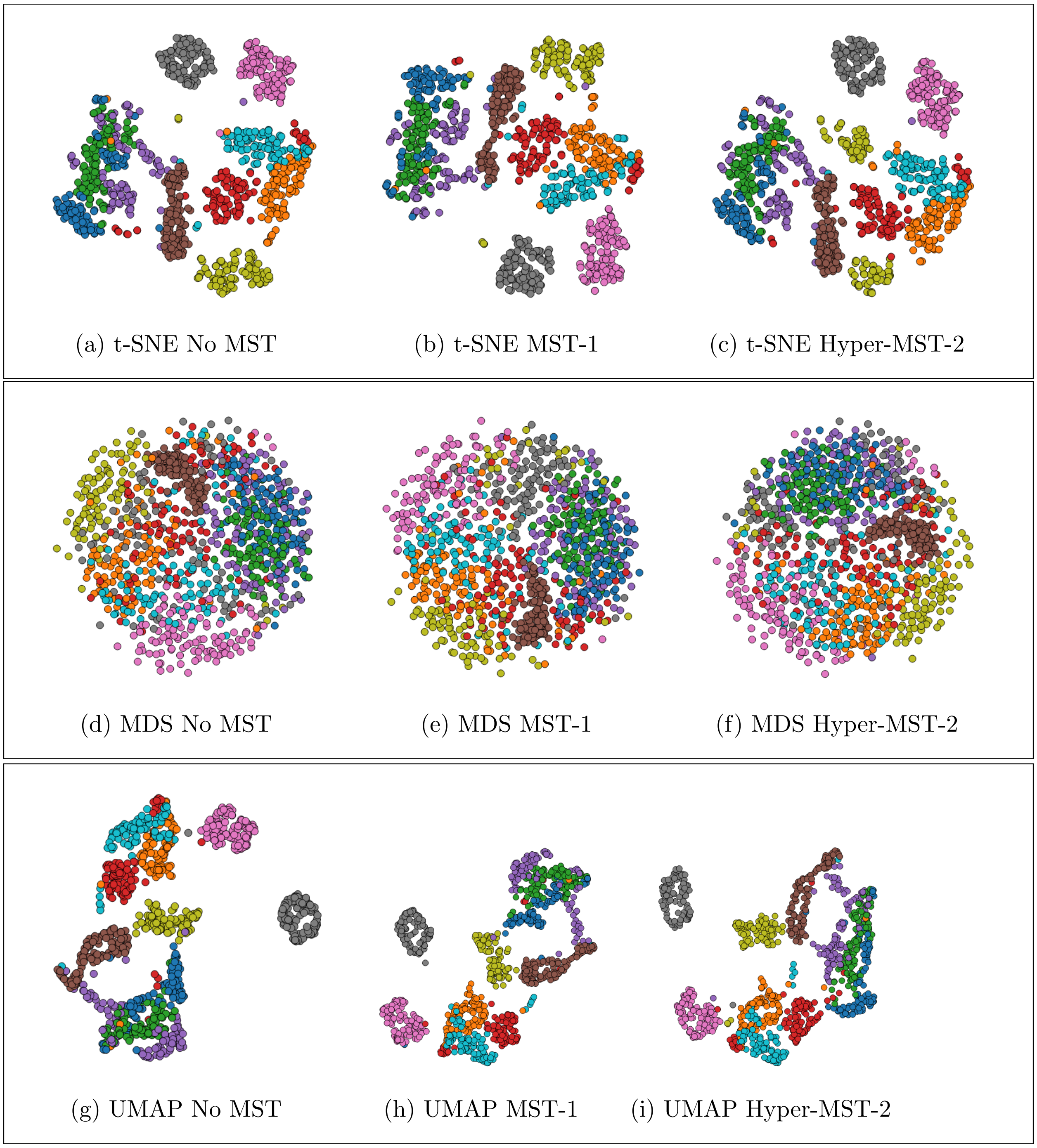


Figure 5.2: Baseline comparisons of dimension reduction algorithms. Each row can be seen as its first (left) instance representing the visualization of 1,000 nodes, without our algorithm applied. The second (middle) instances are the visualizations of the reduced data with the conventional MST applied. The last (right) instances are the result of the data after the hyper-MST-2 was applied to the dimension reduction process. The first row (top) of this figure is t-SNE, the second row (middle) is MDS, and the last row (bottom) is UMAP, applied to an excerpt of MNIST data.

5.3.1 Results

The results of the performance metrics can vary depending on many variables, including the dataset it was performed on. From the results of these baseline tests, we can start to see that our algorithm has some improvements in certain metrics, but can also worsen others. We can also see that there were no drastic changes to the runtime of these experiments with our algorithm implemented. The baseline tables, Tables 5.1, 5.2, and 5.3, show the resulting metric scores of each variation of the algorithm on both MNIST and Iris datasets. The first column of each dataset, where the MST applied is 'none', represents the control algorithm's scores, and the other columns represent the scores when the MST-1 and hyper-MST-2 were applied.

In regards to the MNIST dataset, our algorithm improved structural metrics such as Q_{Local} and Q_{Global} when implemented with t-SNE but caused other scores such as the Spearman rank and Silhouette score to worsen. Our algorithm generally improved the MDS algorithm, minus the DB index score; this indicates that we have improved structural preservation but we have not improved cluster preservation. Not much changed for the UMAP experiment on MNIST, except for slight improvements on Q_{Local}.

Regarding the Iris dataset, our algorithm seems to have a good effect on these baseline experiments, where nearly every metric improved for t-SNE and UMAP, while it

Table 5.2: Baseline MDS metric results. These are the resulting MDS metrics for the baseline experiment.

Baseline Results for MDS

Dataset	MNIST			Iris		
MST Applied	None	MST-1	Hyper-MST-2	None	MST-1	Hyper-MST-2
Spearman	0.6829	0.6777	0.7025	0.9952	0.9961	0.9913
Davies-Bouldin	3.9515	5.0977	5.2036	0.7005	0.6979	0.6957
Silhouette Score	0.01611	0.01877	0.03717	0.5261	0.5276	0.5264
Q_{Local}	0.3800	0.3700	0.3890	0.6361	0.6490	0.6529
Q_{Global}	0.4227	0.4181	0.4416	0.4056	0.4038	0.4053
Runtime (s)	47.529	45.818	46.5313	0.6215	0.6330	0.6231

Table 5.3: Baseline UMAP metric results. These are the resulting UMAP metrics for the baseline experiment.

Baseline Results for UMAP

Dataset	MNIST			Iris		
MST Applied	None	MST-1	Hyper-MST-2	None	MST-1	Hyper-MST-2
Spearman	0.4234	0.3971	0.4196	0.9177	0.9504	0.9420
Davies-Bouldin	2.6337	1.9472	2.8827	0.5607	0.5239	0.5060
Silhouette Score	0.3550	0.3472	0.3523	0.6073	0.6202	0.6431
Q_{Local}	0.4954	0.4957	0.4975	0.5751	0.5719	0.5812
Q_{Global}	0.4743	0.4741	0.4718	0.3473	0.3630	0.3543
Runtime (s)	9.2316	7.7940	7.6135	3.9803	3.3969	3.9650

improved a few metrics for MDS. Not many metrics became worse for this experiment on the Iris dataset, and the MST applications generally improve the visualization and structure performance metrics compared to the control values.

These are initial results without any major modifications such as the number of neighbors used for the KNN graph. Initial impressions lead us to believe that these scores give an indication of improvements caused by the hyper-MST-2 usage, depending on where it is used and whether the user is looking to improve visuals or structural preservation.

Each following experiment will be compared to the control of this baseline to see how further changes to parameters affect the metrics.

5.4 Varying KNN Experiments

The next thing we experiment with is varying the k used for modifying the number of neighbors used to build the KNN graph. We will increase k to see how greater numbers modify how the KNN graph is built, as well as how it affects the resulting dimension reduction algorithm.

Table 5.4: Varying KNN t-SNE metric results. Resulting t-SNE metrics for the experiment of varying k for the KNN graph.

Result for Varying k for t-SNE						
Dataset	MNIST			Iris		
MST Applied	None	Hyper-MST-2		None	Hyper-MST-2	
k Chosen	None	10	20	None	4	5
Spearman	0.5078	0.3957	0.4339	0.9708	0.9760	0.9736
Davies-Bouldin	1.3783	1.8685	1.2818	0.5009	0.5493	0.5298
Silhouette Score	0.3294	0.3042	0.3404	0.6372	0.6107	0.6176
Q_{Local}	0.5368	0.5145	0.5195	0.6509	0.6191	0.6387
Q_{Global}	0.4664	0.4670	0.4821	0.3748	0.3656	0.3668
Runtime (s)	7.2772	7.1838	20.7054	1.0032	1.0033	1.1242

Table 5.5: Varying KNN MDS metric results. Resulting MDS metrics for the experiment of varying k for the KNN graph.

Result for Varying k for MDS						
Dataset	MNIST			Iris		
MST Applied	None	Hyper-MST-2		None	Hyper-MST-2	
k Chosen	None	10	20	None	4	5
Spearman	0.6829	0.6832	0.68837	0.9952	0.9951	0.9951
Davies-Bouldin	3.9515	4.1925	5.5277	0.7005	0.6931	0.6964
Silhouette Score	0.0161	0.0454	0.0100	0.5261	0.5308	0.5259
Q_{Local}	0.3800	0.3868	0.3757	0.6361	0.6445	0.6489
Q_{Global}	0.4227	0.4304	0.4178	0.4056	0.4063	0.4072
Runtime (s)	47.5291	46.6771	59.776	0.6215	0.6325	0.6230

Table 5.6: Varying KNN UMAP metric results. Resulting UMAP metrics for the experiment of varying k for the KNN graph.

Result for Varying k for UMAP						
Dataset	MNIST			Iris		
MST Applied	None	Hyper-MST-2		None	Hyper-MST-2	
k Chosen	None	10	20	None	4	5
Spearman	0.4234	0.4120	0.3956	0.9177	0.9421	0.8233
Davies-Bouldin	2.6337	5.0986	4.1598	0.5607	0.5601	0.5709
Silhouette Score	0.3550	0.3490	0.3505	0.6073	0.6005	0.6033
Q_{Local}	0.4954	0.4991	0.4993	0.5751	0.5745	0.3593
Q_{Global}	0.47425	0.4714	0.4673	0.3473	0.3593	0.3361
Runtime (s)	9.2316	8.4381	22.8248	3.9803	3.8485	4.0247

5.4.1 Results

In Tables 5.4, 5.5, and 5.6, we can see results from varying k for the utilized KNN graph for each experiment. The increase in k is larger for MNIST than for the Iris dataset, in these experiments since MNIST is a larger dataset than Iris.

An increase in k would result in a denser KNN graph, which represents more relationship information in the hyper-dimensional data. We can see that in some instances, an increase in k can benefit some metrics, but a further increase in k can make the same metrics worse. For instance, in Table 5.5, we see that initially, UMAP has an improved Spearman Rank Correlation and Q_{Global} score when $k = 4$. However, when $k = 5$, these scores are diminished. It is likely this is because increasing k is good up to a point, dependent on the data, where we can find good relations, but increasing k further can result in some less important relationships that badly influence the algorithm. We can also see in these tables that a high enough k value can change the runtime of the algorithm drastically; even more so with a greater node range.

On the MNIST dataset, increasing k seemed to increase clustering metrics such as the DB index, Silhouette score, and Q_{Global} value with t-SNE, and the Q_{Local} score with UMAP. Some metrics became worse, however, specifically the DB index and Spearman Rank Correlation values with MDS and UMAP. In the Iris dataset, the t-SNE metrics diminished except for the Spearman Rank correlation value, which indicates good distance preservation throughout the dimension reduction process. However, the diminishing other scores indicate a lack of quality cluster preservation. The Iris dataset did have better results with MDS, where almost every metric improved. The UMAP technique also improved for $k = 4$, but declined when k was set to 5.

Overall, a higher k value can be seen to improve some metrics, but can also worsen others. It is also seen that a k should be carefully chosen, as there is a fine line between a value too small and a value too high, in order to capture the right amount of information to be effective in this algorithm.

5.5 Barcode Thresholding Experiments

The next experiment concerns varying the thresholding range for the hyper-MST-2 application. The barcode threshold allows the users to choose which hyper-edges to apply from the hyper-MST-2 to the dissimilarity graph. These components are sorted by their importance to the graph, where a threshold range 'r' in a dataset means the first 'r' most important components of the MST that is used for the dimension reduction influencing process. Any edges outside this threshold would not be applied.

Table 5.7: Varying threshold t-SNE metric results. Resulting t-SNE metrics for the experiment of varying the threshold range for components.

Result for Varying Thresholds for t-SNE						
Dataset	MNIST (1000 Nodes)			Iris (150 Nodes)		
MST Applied	None	Hyper-MST-2		None	Hyper-MST-2	
k Chosen	None	10		None	4	
Threshold Range (r)	999	499	49	149	75	50
Spearman	0.5078	0.4454	0.4852	0.9708	0.9713	0.9766
Davies-Bouldin	1.3783	1.3230	1.5142	0.5009	0.5254	0.5045
Silhouette Score	0.3293	0.3239	0.3210	0.6372	0.6225	0.6348
Q_{Local}	0.5368	0.5215	0.5165	0.6509	0.6585	0.6460
Q_{Global}	0.4664	0.4797	0.4781	0.3748	0.3690	0.3664

Table 5.8: Varying threshold MDS metric results. Resulting MDS metrics for the experiment of varying the threshold range for components.

Result for Varying Thresholds for MDS						
Dataset	MNIST (1000 Nodes)			Iris (150 Nodes)		
MST Applied	None	Hyper-MST-2		None	Hyper-MST-2	
k Chosen	None	10		None	4	
Threshold Range (r)	999	499	49	149	75	50
Spearman	0.6829	0.6653	0.5358	0.9952	0.9941	0.9973
Davies-Bouldin	3.9515	5.3577	4.8654	0.7005	0.6897	0.7008
Silhouette Score	0.0161	0.0206	0.0246	0.5261	0.5247	0.5254
Q_{Local}	0.3800	0.3741	0.3889	0.0.6361	0.6463	0.6468
Q_{Global}	0.4227	0.4174	0.4374	0.4056	0.4016	0.4067

Table 5.9: Varying threshold UMAP metric results. Resulting UMAP metrics for the experiment of varying the threshold range for components.

Result for Varying Thresholds for UMAP						
Dataset	MNIST (1000 Nodes)			Iris (150 Nodes)		
MST Applied	None	Hyper-MST-2		None	Hyper-MST-2	
k Chosen	None	10		None	4	
Threshold Range (r)	999	499	49	149	75	50
Spearman	0.4234	0.4076	0.4577	0.9177	0.9390	0.9478
Davies-Bouldin	2.6337	3.018	4.6774	0.5607	0.5596	0.5074
Silhouette Score	0.3550	0.3645	0.3524	0.6073	0.6262	0.5765
Q_{Local}	0.4954	0.4951	0.4997	0.5751	0.5739	0.5765
Q_{Global}	0.47425	0.4702	0.4675	0.3473	0.3648	0.3487

5.5.1 Results

In Tables 5.7, 5.8, and 5.9, we can see results from varying the threshold range for the utilized hyper-MST-2 graph for each experiment. The threshold range selected is smaller in the Iris test than for MNIST since these are chosen in accordance with the size of each dataset.

Selecting a smaller range within these tables means the only components from the MST that were applied were the most important ones, rather than being applied along with the following, lesser-important components.

With both the MNIST and Iris datasets, selecting a lower threshold range seemed to improve structural scores, such as the Spearman Rank Correlation and coranking Q scores when the threshold range was smaller, than when it was larger. Generally, the more selective and smaller thresholding ranges improved such structural metrics, especially when used with MDS and UMAP. However, such a smaller range seemed to have an inverse effect on the clustering metrics, namely the DB index and Silhouette scores.

Chapter 6: Discussion and Conclusion

In this thesis, we have presented an algorithm to improve structural preservation in currently existing dimension reduction techniques, specifically t-SNE, MDS, and UMAP. These algorithms have been shown to be robust techniques in their own rights for dimension reduction. Though it is challenging, our introduced algorithm offers some improvements as seen in our experiments, in regards to our selected structure and cluster preserving metrics.

6.1 Results Discussion

In many of the experiments, the results show that our algorithm does well to improve coranking Q scores, and thus improves the preservation of the structure, albeit not as drastically as expected. Our algorithm also varies in its effect on the other metrics such as the Spearman rank correlation, DB index, and Silhouette scores; sometimes improving and sometimes worsening these metrics. From our experimental results, it also appears that our algorithm fares better with the smaller Iris dataset, though this could be dependent on the parameters selected.

At this point, there is not much to be generalized about our algorithm from these results, since the results are indeed mixed depending on the data and parameters used. However, in some instances where the overall change was for better or worse, we can see that our implementation and program allow the user to directly affect the dimension reduction algorithms to an extent, giving them a finer-grain control over the process.

6.2 Limitations

Throughout the experiments, we can often see that when some scores improve, others decline. One limitation of our algorithm is that there may be some trade-off between our metrics, mainly when the change of the structural metrics are compared to those that measure the clusters of the resulting data.

It is possible that some of these metrics could be optimized within our algorithm, in order to improve some of the metrics related to structural preservation or clustering, after the dimension reduction is complete. It is important to remember that while these metrics are often used for measuring the quality of dimension reduction and to give us some insight into how well the algorithm does in these regards, the metrics are not a direct way to measure performance. Due to the apparent trade-off of cluster and structural preservation seen from our metrics, an application of our algorithm should specify which they intend to improve, before optimizing it to one metric or another.

6.3 Future Work

6.3.1 Expanding Upon the Hyper-MST

The hyper-MST-2 discussed and used in this thesis was of clique-size three. These captured the relationship between three different nodes in the hyper-dimensional space, rather than two relationships that a conventional MST would capture. The hyper-MST could be taken further to create a hyper-MST-3, hyper-MST-4, and even some hyper-MST-N. This would capture more information about the relationships of the data and hopefully pass that information to the dimension reduction process to further improve the resulting data.

There are also future experiments and optimizations to be made for our algorithm. For one, the modifications applied to the dissimilarity matrix was through a linear mapping which emphasized the more important components of the hyper-MST-2 differently than the lesser important ones. This could be changed to affect the dissimilarity matrix differently to have an emphasis on other structures.

Even though some experiments seem to positively affect the dimension reduction process, further research is still needed to ensure that the structures we are emphasizing with our algorithm are indeed structures in the data. We have seen improvements to some extent, but these experiments were limited to certain datasets, node ranges, and other constraints. Further research is suggested in order to ensure that what we have observed as improvements in regards to the resulting performance metrics, do indeed refer to emphasized structures within the hyper-dimensional data.

References

[1] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[2] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011.

[3] Jong Youl Choi, Seung-Hee Bae, Xiaohong Qiu, and Geoffrey Fox. High performance dimension reduction and visualization for large high-dimensional data analysis. pages 331–340, 2010.

[4] Michael AA Cox and Trevor F Cox. Multidimensional scaling. In *Handbook of data visualization*, pages 315–347. Springer, 2008.

[5] David L Davies and Donald W Bouldin. A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence*, (2):224–227, 1979.

[6] Daniel Engel, Lars Hüttenberger, and Bernd Hamann. A Survey of Dimension Reduction Methods for High-dimensional Data Analysis and Visualization. 27:135–149, 2012.

[7] Hadi Fanaee-T and Magne Thoresen. Performance evaluation of methods for integrative dimension reduction. *Information Sciences*, 493:105–119, 2019.

[8] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[9] Paulo Joia, Danilo Coimbra, Jose A Cuminato, Fernando V Paulovich, and Luis G Nonato. Local affine multidimensional projection. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2563–2571, 2011.

[10] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

[11] Mark Joswiak, You Peng, Ivan Castillo, and Leo H Chiang. Dimensionality reduction for visualizing industrial chemical process data. *Control Engineering Practice*, 93:104189, 2019.

[12] Sung-Soo Kim, Sunhee Kwon, and Dianne Cook. Interactive visualization of hierarchical clusters using mds and mst. *Metrika*, 51(1):39–51, 2000.

[13] Dmitry Kobak and George Linderman. Initialization is critical for preserving global data structure in both t-sne and umap. *Nature Biotechnology*, 39:156–157, 2021.

[14] Tomasz Konopka and Maintainer Tomasz Konopka. R-package: umap. *Uniform Manifold Approximation and Projection*, 2018.

[15] Guido Kraemer, Markus Reichstein, and Miguel D Mahecha. dimred and coranking—unifying dimensionality reduction in r. *R Journal*, 10(1):342–358, 2018.

[16] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[17] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[18] John A Lee and Michel Verleysen. Quality assessment of dimensionality reduction: Rank-based criteria. *Neurocomputing*, 72(7-9):1431–1443, 2009.

[19] Laurens Van Der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:25792605, 2008.

[20] Laurens Van Der Maaten and Geoffrey Hinton. Visualizing non-metric similarities in multiple maps. *Machine Learning*, 87(1):33–55, 2011.

[21] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[22] Safa A Najim and Ik Soo Lim. Trustworthy dimension reduction for visualization different data sets. *Information Sciences*, 278:206–220, 2014.

[23] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[24] Robert Clay Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.

[25] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[26] Ashley Suh, Mustafa Hajij, Bei Wang, Carlos Scheidegger, and Paul Rosen. Persistent homology guided force-directed graph layouts. *IEEE Transactions on Visualization and Computer Graphics*, page 1–1, 2019.

[27] Flora S. Tsai. A visualization metric for dimensionality reduction. *Expert Systems with Applications*, 39(2):1747–1752, 2012.

[28] Dongfang Wang and Jin Gu. Vasc: dimension reduction and visualization of single-cell rna-seq data by deep variational autoencoder. *Genomics, proteomics & bioinformatics*, 16(5):320–331, 2018.

[29] Christopher KI Williams. On a connection between kernel pca and metric multidimensional scaling. *Machine Learning*, 46(1):11–19, 2002.