Hierarchical Landmark Charting

Suamporn Ketpreechasawat
Department of Computer Science
Brown University
Providence, RI, USA 02912-1910
suamporn@cs.brown.edu

Advisor:
Odest Chadwicke Jenkins
Department of Computer Science
Brown University
Providence, RI, USA 02912-1910
cjenkins@cs.brown.edu

Abstract

In this paper, we propose Hierarchical Landmark Charting (HLC) for discovering low-dimensional structures underlying high-dimensional data sets with large numbers of data objects. The first step of the algorithm is to partition the input data in a multi-resolution based on the geodesic distances between the points. Each data partition can be embedded individually by classical multidimensional scaling (MDS) algorithm and then aligned into a single embedding. The HLC algorithm has an advantage over Landmark MDS (LMDS) in term of storage since its memory requirements are independent of the number of data sets. We demonstrate HLC by applying it to various Swiss roll data sets ranging between 500 and 500,000 points.

1 Introduction

Recently there has been an increase of interest in non-linear dimensionality reduction. These days, advancements in sensor and embedded technologies can provide us with not only a large amount of data, but also a large amount of high dimensionality such as fingerprint recognition, gene distributions, or human gestures. Working with these problems often involves working with large volumes of high-dimensional data, which leads to the creation of very large databases, and sometimes it is hard to extract the knowledge contained in these databases. Fortunately, these high-dimensional data sets contain some latent low-dimensional structures that are usually more efficient and can be processed by other learning algorithms.

Recent developments in the area of manifold learning have focused on algorithms for uncovering these low-dimensional representations on non-linear data. One approach is local or shape preserving that preserves local geometry of data (e.g., LLE [6], Hessian LLE [4]).

Another approach is global or topology preserving that preserves the pair-wise relationships of data (e.g., Isomap [1], SDE [5]). Unfortunately, these algorithms are often limited by the number of input points N due to the expensive storage of and operations on a $N \times N$ matrix. Generally, these methods are limited in their applicability to large data sets with more than hundreds of thousands of data objects.

Recently, several approaches have been proposed to overcome those limitations; in other words, they propose how to discover a low-dimensional structure lying on a large, high-dimensional input space while preserving all of its attractive properties (e.g., LMDS [3], PE [7], Out-of-sample embedding [8]). In this paper, we propose an alternative method for performing non-linear dimensionality reduction on large data sets, which we call *Hierarchical Landmark Charting* (HLC). The HLC method generates a hierarchical data structure of input data into *meta-neighborhoods* based on the geodesic distances between the points. Each *meta-neighborhood* is separately embedded by classical MDS and merged into a single embedding using *global landmarks*, which are the roots of the hierarchical data structure.

In the next section, we look at related work in dimensional reduction. In sections 3 and 4, we describe the algorithm of HLC. In section 5, we present the complexity and analysis for HLC. Lastly, in section 6, we run HLC on generated Swiss roll data sets ranging between 500 and 500,000 points, and make comparisons to LMDS. HLC is able to embed these data sets efficiently, which produces an accurate embedding qualities within a reasonable running time.

2 Related Works

There are several approaches to discovering a low-dimensional data structure lying within high-dimensional data. Isomap [1] generalizes classical MDS to non-linear manifolds. It uses major algorithms in classical MDS with geodesic distances which can capture non-linear structures between points instead of simply taking Euclidean distances. It first determines the nearest neighbors of each point in the data set and represents them as a weigh graph. Then, it computes the pair-wise distance matrix between the points as the shortest patch distance in a weigh graph. Finally, it uses classical MDS to that matrix to produce an embedding. This approach works with global pair-wise distances and is able to produce an embedding that preserves the high dimensional manifold's estimated intrinsic geometry and can compute a globally optimal solution. LLE [6] can be considered a local alternative to the global reduction of Isomap. It attempts to maintain only local geometry, and is therefore less stable and less computationally expensive. This approach determines the nearest neighborhoods and then computes the weights of local neighborhoods associated with a point. Then, it uses that information to construct the embedding. This method also allows for the discovery of the intrinsic geometry of the high dimensional manifold and avoids a local minimum. However, these approaches are not suited for large data sets due to the expensive operations on a $N \times N$ matrix.

With the increasing availability of very large data sets, scalable dimensionality reduction algorithms have become more important. Various algorithms have been proposed to deal with the large volume of data sets. Landmark MDS [3] is a combination of the global geometric approach (Isomap [1]) and local approach (LLE [6]). The results produce computational efficiency as well as greater stability. It chooses a subset of data called "landmark points" and embeds the landmark points using classical MDS. Then, it determines the position of remaining points by using the distances to the already-embedded landmark points. This approach is largely dependent on landmark selection and is therefore sensitive to noise. Out-of-sample embedding [8] is an extension of MDS, spectral clustering, Laplacian eigenmaps, Isomap and LLE. It allows a trained model to be applied to out-of-sample points without recomputing eigenvectors. For Parametric Embedding [7], besides

data points, the algorithm needs to know the data points' classes and the probabilities that data points are related to each class. It relates the distance between that data and class to some probability in order to construct the low dimensional structure. It performs unsupervised dimensionality reduction. Although it has several advantages over other methods for embedding a single set of data points based on their pair-wise relationship, a set of data is required to be classified and it can be applied efficiently to large data sets only if the number of classes is small.

Unlike PE, HLC does not need a probabilistic model; it partitions data sets into local neighborhoods similar to LLE. Instead of embedding only landmark points and finding the coordinates of the remaining points using the geodesic distances as in LMDS, HLC embeds each local neighborhood with global landmark points separately, similar to Isomap, and then uses the coordinates of global landmark points to find the coordinates of the final embedding.

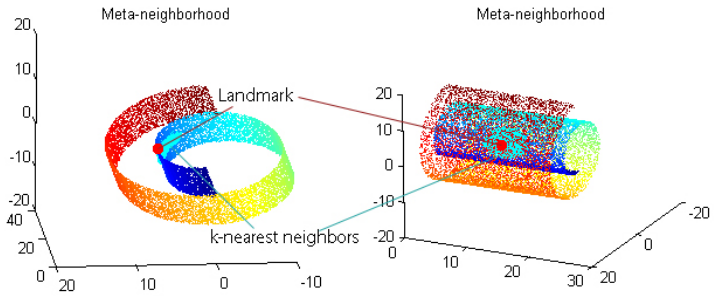


Figure 1: Randomly choose the landmark point and construct the meta-neighborhood of size k which is the k nearest points to the landmark point. A landmark is displayed in red point and meta-neighborhood is displayed in cyan on the original data.

3 Constructing Hierarchical Landmark Charts

The HLC algorithm performs two primary steps: 1) Hierarchically partitions data into meta-neighborhoods or charts which are used to construct the structure called *Hierarchical Landmark Charts*, and 2) individually embeds each meta-neighborhood using classical MDS; all embeddings are aligned into a single embedding by using the global landmark, which is the top level of the hierarchical structure. In this section, we describe the hierarchical partitioning step.

Given a set of N data points X, $\{x_i | x_i \in X \wedge 1 \leq i \leq N\}$ that we wish to embed and its global distance function $g_x(x_i, x_j)$, which specifies the Euclidian distance between a point x_i to point x_j in X, the hierarchical partition step starts with constructing a sparse matrix A of neighborhood edges.

The procedure constructs a fine-to-coarse partitioning of the input data into meta-neighborhoods as illustrated by figure 1. Beginning with the input data X, the finest resolution of meta-neighborhoods, or chart level 1 C_1, is computed in the following fashion. The first landmark l_1,i is randomly selected from the input data and Dijkstra's algorithm is used to compute the geodesic distance matrix Di, which is the $N \times 1$ matrix composed of the distances from the landmark $l_{1,i}$ to input data X and the distance matrix is kept in storage so it can be reloaded without re-computing. The distance matrix D_i is sorted and

the k nearest points of the landmark $l_{1,i}$ are used to construct the meta-neighborhood $M_{1,i}$. For each iteration, we randomly select the landmark from previously points that were not selected and are unassociated with the meta-neighborhoods points, and construct the meta-neighborhood using the same procedure. It stops when there are no more available points.

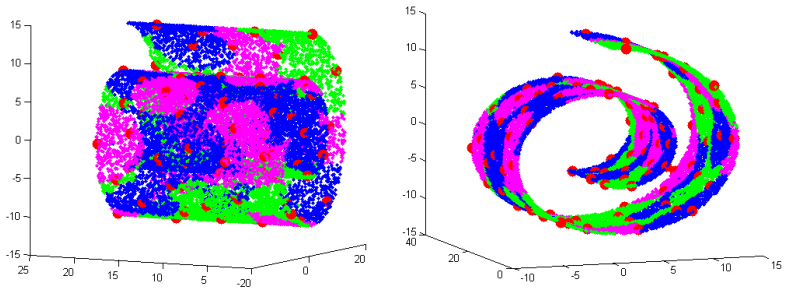


Figure 2: Hierarchical partitioning. Begin with the input data; we construct the set of meta-neighborhoods in the fine-to-coarse fashion. The global landmarks are displayed in red and data partitions are displayed in blue green and magenta.

For the coarser levels, we used the same procedure as the finest level, but with the previous level's landmark points set L_1, $\{l_{1,i}|l_{1,i} \in L_1\}$ instead of the input data X. The distance matrix can be loaded from storage to avoid repeat computations with Dijkstra's algorithm. The original data is hierarchical partitioned as in Figure 2.

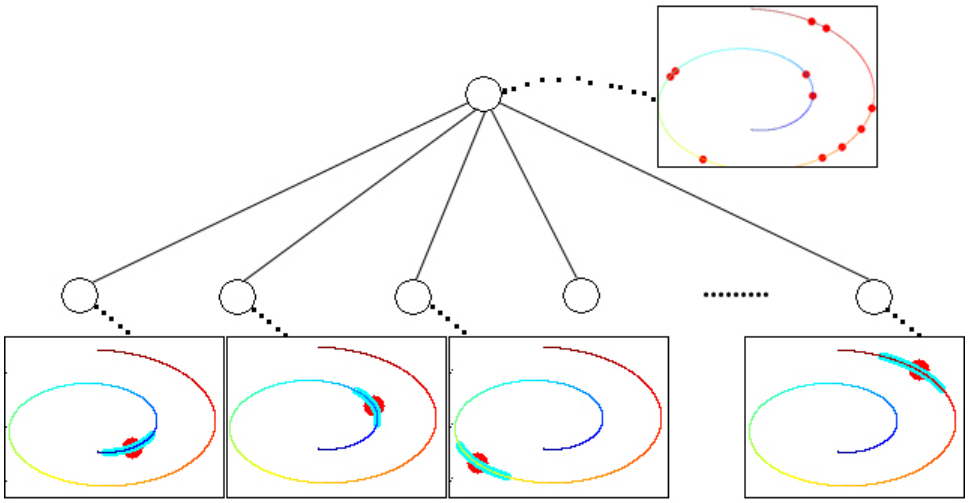


Figure 3: The two level's tree structure for HLC. Leaves are finest partitions of original data. Each leaf displays a meta-neighborhood in cyan and its landmark in red. A root is the coarsest partition of original data. The global landmarks are display in red spots.

The hierarchical partition continues coarsening until the number of landmark points L_J falls below the meta-neighborhood's size k. The result from the partitioning is a tree structure in which each node $C_{j,i}$ is associated with a set of points $M_{j,i}$ and a landmark point

$l_{j,i}$, indicating its parent as illustrated by figure 3. The root node C_J contains a set of landmark points L_J from the coarsest level, considered the *global landmarks* L of the data.

4 Embedding Hierarchical Landmark Charts

An embedding is performed from the root to the leaf of the tree, in a coarse-to-fine manner. We embedded a set of points in each node of the tree together with the global landmark points, which are a set of points at the root of the tree. Beginning with the root, the global landmarks $M_{J,1} = L$ were embedded using Classical MDS to form the global embedding Y_L, which were used later as a template when all embeddings were aligned into a single embedding as illustrated by figure 4. The $|L| \times |L|$ distance matrix $G_L(L)$ that stores the distances between the global landmark points can be constructed by loading the pre-computed distance matrix. Continuing on the node $M_{j,i}$ at a finer level j, the points associated with this node were embedded using Classical MDS on the following block-partitioned matrix to form a $Y_{j,i}$.

$$\left[\begin{array}{c|c} G_{M_{j,i}}(M_{j,i}) & G_L(M_{j,i}) \\ \hline G_{M_{j,i}}(L) & G_L(L) \end{array}\right] \tag{1}$$

where $G_{M_{j,i}}(M_{j,i})$ is a pair-wise distance matrix among the points of the node $M_{j,i}$. $G_L(M_{j,i})$ and $G_{M_{j,i}}(L)$ are distance matrices between the global landmarks and the points of the node $M_{j,i}$. Usually, we can construct the distance matrix by loading the pre-computed distance matrix - except the $G_{M_{1,i}}(M_{1,i})$ which is the distance matrix of the finest level since it may contain a set of points which we have never computed the distance matrix. We constructed the distance matrix $G_{M_{1,i}}(M_{1,i})$ using Dijkstra's algorithm to compute the distances between the points of the node $M_{1,i}$, and continued to use the same procedure with all of the remaining nodes.

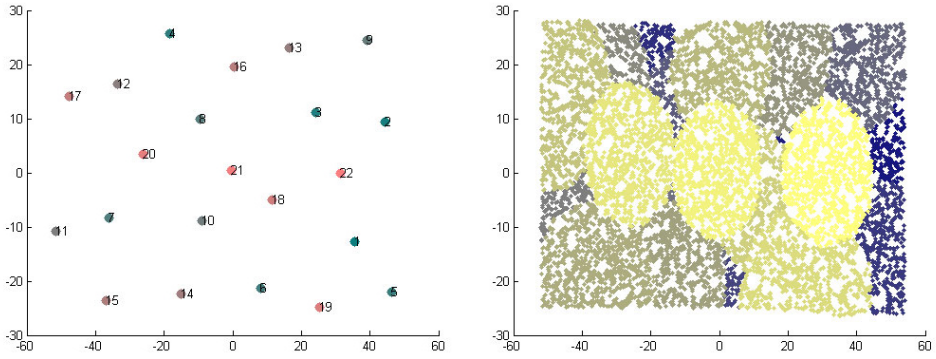


Figure 4: The embedding of global landmarks is displayed on the left and the embedding of finest partitions is displayed on the right.

An alignment was also performed from the root to the leaf of the tree by using an affine transformation. The global embedding or currently solved points Y_L, was used as a template to solve for the transformation of the remaining embeddings. For each embedding $Y_{j,i}$, we found the overlap between this embedding and all the solved points. The overlapping points are usually the global landmarks and some of the other points we have already solved for the transformation that appear in the current embedding $Y_{j,i}$. To solve the trans-

formation for embedding dimensionality d, let $Coords$ be the $d \times N$ matrix of solved points and t be the number of overlapping points. We use the follow equations:

$$GC = Coords(Coords \cap Y_{j,i}) \tag{2}$$

$$LC = Y_{j,i}(Coords \cap Y_{j,i}) \tag{3}$$

$$LC = U\Sigma V^T \tag{4}$$

$$LC^+ = V\Sigma^{-1}U^T \tag{5}$$

$$trans = GC \times LC^+ \tag{6}$$

where GC is the $d \times t$ matrix whose rows are composed of d-dimensional coordinates of solved points that overlap with the current embedding, and LC is the $d \times t$ matrix whose rows are composed of d-dimensional coordinates of current embedding that overlap with the solved points. Equation 4 shows the SVD decomposition of LC, and equation 5 shows the pseudo-inverse of LC (LC^+). Then, the transformation of LC from GC can be computed by using equation 6.

The true embedding coordinate of meta-neighborhood $M_{j,i}$ can be estimated by using follow equations:

$$real = trans \times B \tag{7}$$

$$B = \begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,|M_{j,i}|} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,|M_{j,i}|} \\ \vdots & \vdots & \ldots & \vdots \\ y_{d,1} & y_{d,2} & \cdots & y_{d,|M_{j,i}|} \\ 1 & 1 & \ldots & 1 \end{bmatrix} \tag{8}$$

where $y_{l,m}$, $l = 1 \ldots d$ and $m = 1 \ldots |M_{j,i}|$ are the l-coordinate of point m of embedding $Y_{j,i}$. $real$ is the true embedding coordinates of $M_{j,i}$. All nodes at the leaves are solved for real coordinates. An illustration of equations 2 to 7 is displayed in figure 5. Pseudocode for the algorithm is given in the Appendix.

5 Analysis

As stated by de Silva and Tenenbaum [2], Isomap requires $O(N^2)$ memory and $O(N^3)$ computation, where N is the number of input points. LMDS significantly decreases these requirements to $O(nN)$ memory and $O(dnN + n^3)$ computation, where n and d are the chosen number of landmarks and embedding dimensionality respectively.

For our HLC implementation, it requires $O(nN)$ storage, $O(N)$ memory, and $O(nN^2)$ computation in the hierarchical partitions step. We save global distances from each landmark we selected $l_{j,i}$ in file storage and load only when necessary, which requires $O(nN)$ storage where n is the number of landmarks we selected in the finest level. Since it requires loading a single file for each landmark, the memory requirememt is $O(N)$. For the computation time, HLC uses Dijkstra's algorithm to compute the $N \times 1$ distance matrix for selected landmarks which takes $O(N^2)$ computation and since we have n landmarks for the finest level, the first iteration takes $O(nN^2)$ computation. The number of iterations is dependent on the size of the chart c; normally the HLC gives us 2-3 levels in the tree and we can simply load the distance matrix we computed in the finest level so that the computation time of this step is dependent largely on the finest level partition which yield $O(nN^2)$ computation.

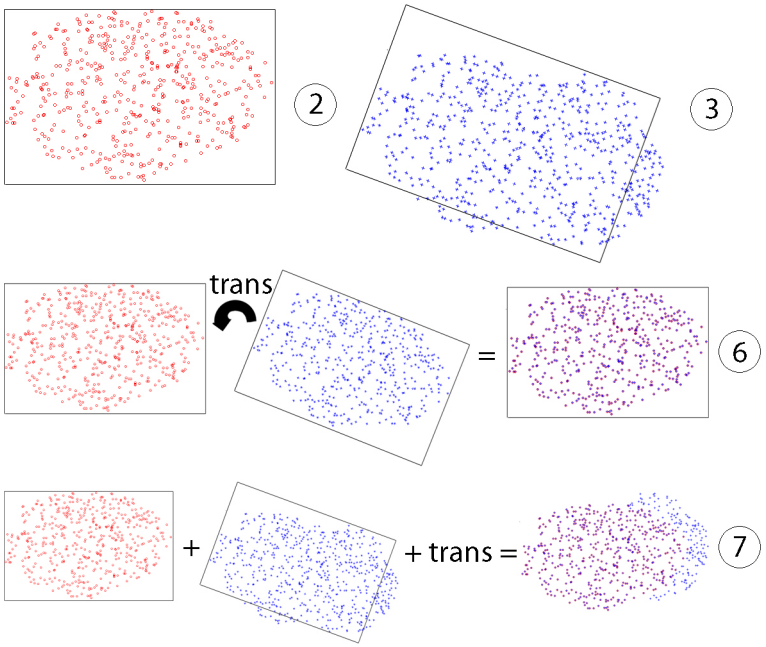


Figure 5: The diagram for equations 2 - 7.

For the embedding step with the chart's size c, HLC requires $O(c^2)$ memory and $O(nc^3)$ computation, where $|L|$ is the size of global landmarks. The memory requirement is dependent on the size of the matrix we embedded and the maximum size of the meta-neighborhood matrix is equal to the chart's size c plus the number of global landmarks that are bounded by $O(c)$, so the embedding step requires $O(c^2)$ amount of memory. The computational time is dependent largely on the embedding operation on the meta-neighborhood matrix and Dijkstra's algorithm on the meta-neighborhood matrix. The embedding operation takes $O(c^2)$ computation and Dijkstra's algorithm takes $O(c^3)$ on $(c + |L|) \times (c + |L|)$ matrix. This step iterates for n times on the finest level, which yields the total of $O(nc^3)$ computation. The total requirements of HLC are $O(nN^2 + nc^3)$ computation and $O(N + c^2)$ memory.

The HLC memory requirements for embedding are independent of N and can be adjusted by the chart size. The advantage of HLC over LMDS is not necessarily in faster computation, but in usage of the storage to scale with increasing N. LMDS does not work well when the data sets are very large and we cannot use the reasonable number of landmark points, while on HLC we can adjust the size of the chart.

6 Results

In this section, we describe two results with HLC implementation applied to synthetically-generated Swiss roll data sets ranging between 500 and 500,000 points.

6.1 Swiss roll without noise

HLC performed equivalent to Landmark MDS when dealing with large data sets without noise. In this section we applied Isomap, HLLE, LMDS, and HLC to Swiss roll data sets

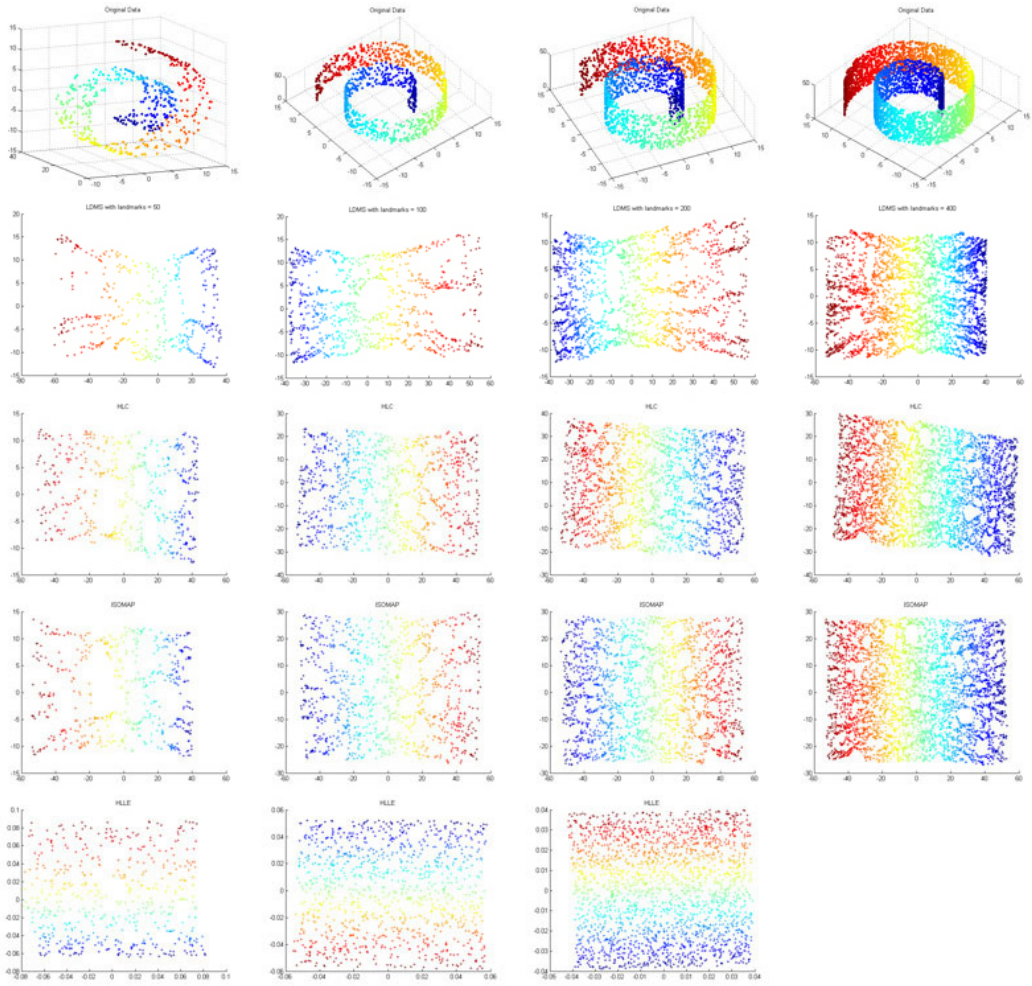


Figure 6: HLC is applied to a Swiss roll data set without noises. The first row is the original data set of 500, 1000, 2000 and 4000 points. The second row is the embedding generated by LMDS with 10% landmark points. The third row is the embedding generated by HLC with chart size = 50, 50, 100 and 1000. The fourth row is the embedding generated by ISOMAP. The fifth row is the embedding generated by HLLE.

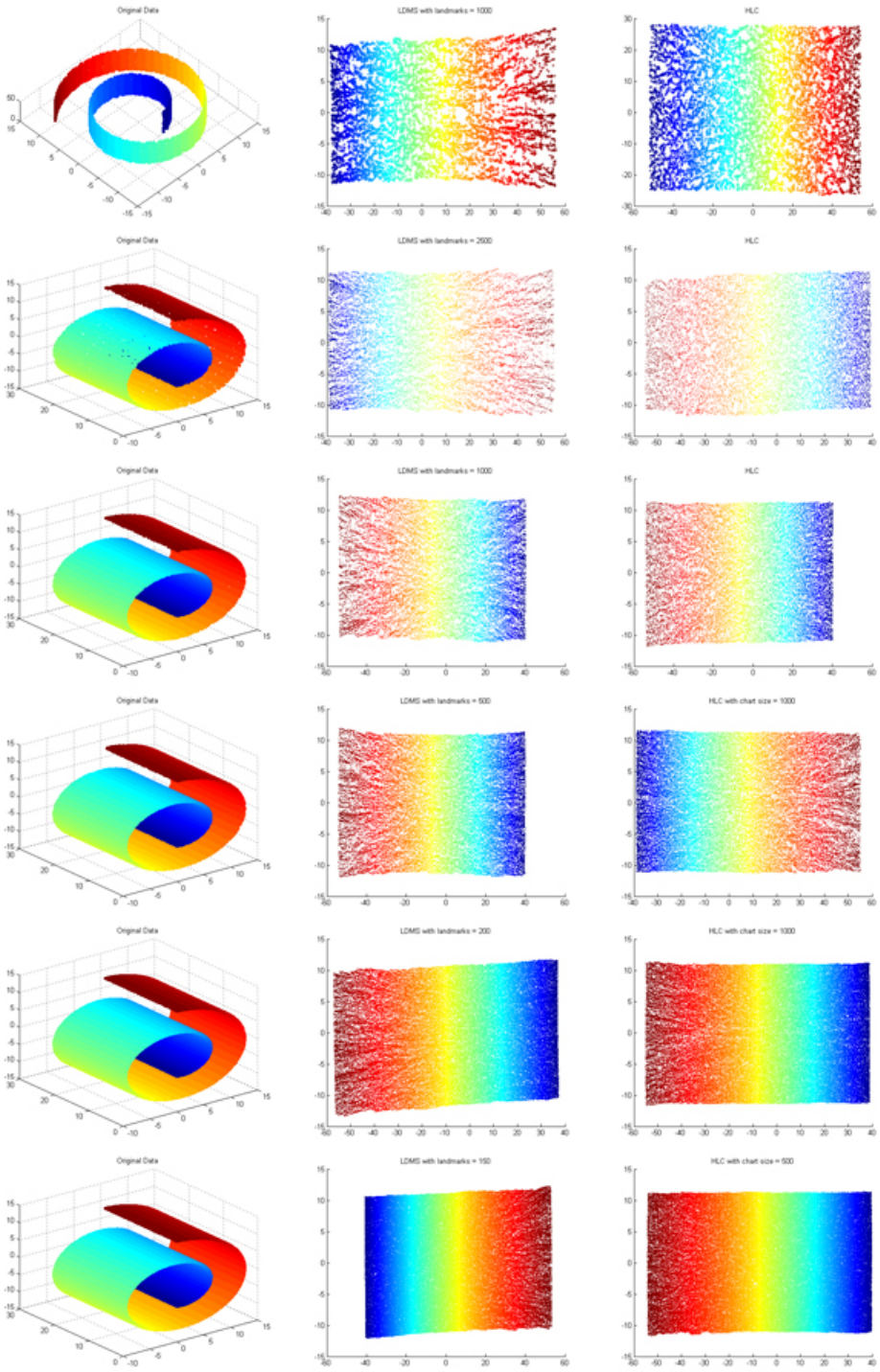


Figure 7: HLC is applied to a Swiss roll data set without noises. The first column is the original data set of 10k, 25k, 50k, 100k, 300k and 500k points. The second column is the embedding generated by LMDS with 1000, 2500, 1000, 500, 200 and 150 landmark points. The third column is the embedding generated by HLC with chart size = 1000 and 500.

ranging from 500 to 500,000 points. We found that it is considerably cheaper usage of the memory to run HLC than to run LMDS.

As illustrated in figure 6, we applied Isomap, HLLE, LMDS (with 10% randomly selected landmarks), and HLC (with chart size of 50, 50, 100, and 1000) to the Swiss roll data sets consist of 500, 1000, 2000, and 4000 data points. It is apparent that all algorithms gave us equivalent embedding quality.

Figure 7 shows the same data sets consist of 10,000, 25,000, 50,000, 100,000, 300,000, and 500,000 data points for which we applied LMDS (with 1000, 2500, 1000, 500, 200, and 150 randomly selected landmarks), and HLC (with chart size of 1000, 1000, 1000, 1000, 1000, and 500). Since the data sets were too large we could not apply both Isomap and HLLE to these data sets due to the cost of the operations. The quality of embeddings between the two algorithms is slightly different. However, the overall shape of the embedded data produced by HLC and LMDS is equivalent.

To illustrate that HLC and LMDS are equivalent, we considered a mean distance error of embedding data. Figure 8 shows the graphs that display a mean distance error produced by HLC and LMDS. The lower the mean distance error, the better the embedding quality, because the low value of a mean distance error means that there is a small difference between truth embedding and embedding produced by an algorithm. It was observed from the graph that mean distance errors of embedding data produced by HLC are slightly lower than those of LMDS. The difference is not significant so that two methods are equivalent.

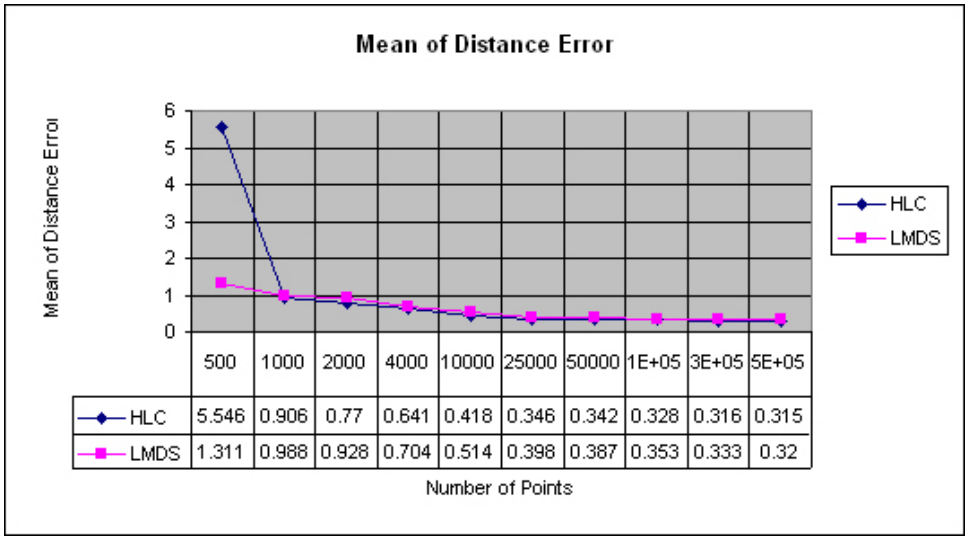
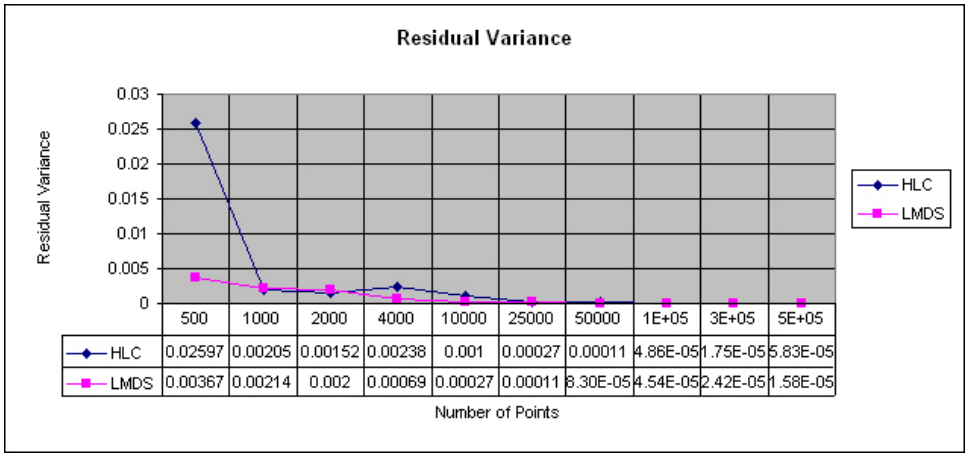


Figure 8: A Mean Distance Error of LMDS vs HLC running on Swiss roll without noise.

Residual variances of embedding data produced by HLC were slightly higher than those of LMDS, as illustrated by figure 9. However, this will not have a significant effect on the embedding quality since the difference is relatively small.

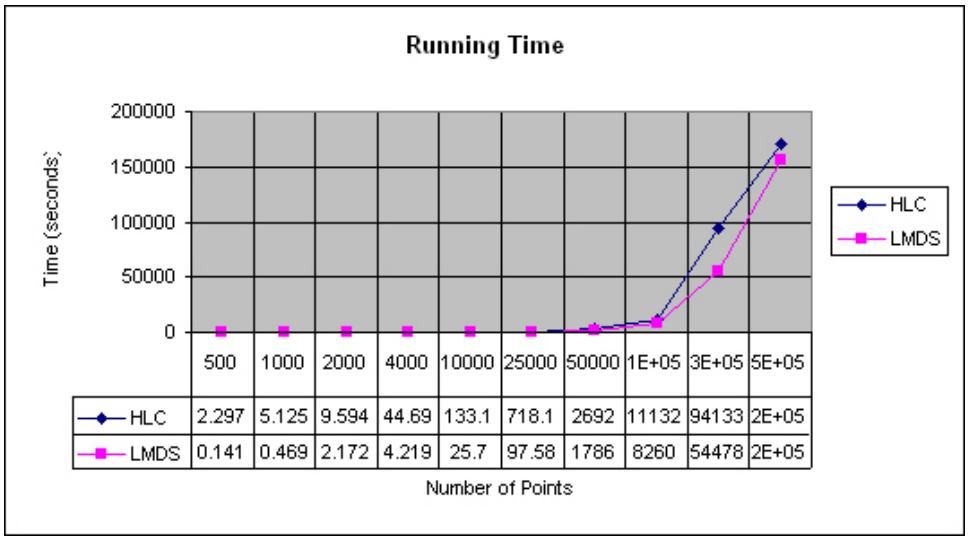
Figure 10 shows the graphs that display the running time (in seconds) required by HLC and LMDS. It can be observed from the graph that the running time increases as the number of input points increases, and the running times of both algorithms increase significantly when the number of input points is very large (100,000 to 500,000 points). The differences between the running times required by HLC and LMDS are significant when the number of input points is small. This is because the overhead occurred by means of the partition step,



	500	1000	2000	4000	10000	25000	50000	1E+05	3E+05	5E+05
HLC	0.02597	0.00205	0.00152	0.00238	0.001	0.00027	0.00011	4.86E-05	1.75E-05	5.83E-05
LMDS	0.00367	0.00214	0.002	0.00069	0.00027	0.00011	8.30E-05	4.54E-05	2.42E-05	1.58E-05

Figure 9: Residual Variances of LMDS vs HLC running on Swiss roll without noise.

which does not have much effect on the large data sets. However, HLC requires a longer running time than LMDS.



	500	1000	2000	4000	10000	25000	50000	1E+05	3E+05	5E+05
HLC	2.297	5.125	9.594	44.69	133.1	718.1	2692	11132	94133	2E+05
LMDS	0.141	0.469	2.172	4.219	25.7	97.58	1786	8260	54478	2E+05

Figure 10: A running time (seconds) of LMDS vs HLC running on Swiss roll without noise.

For the memory requirement, HLC needs a lower memory than LMDS, as illustrated in figure 11. The difference in memory requirements between HLC and LMDS is relatively miniscule when the data sets are small. However, when the data becomes large the difference between the two algorithms becomes significant as well, owing to an expensive operation on an embedding step of LMDS. In contrast to LMDS, HLC keep data in storage and will be loaded into memory later as needed, which significantly influences the performance.

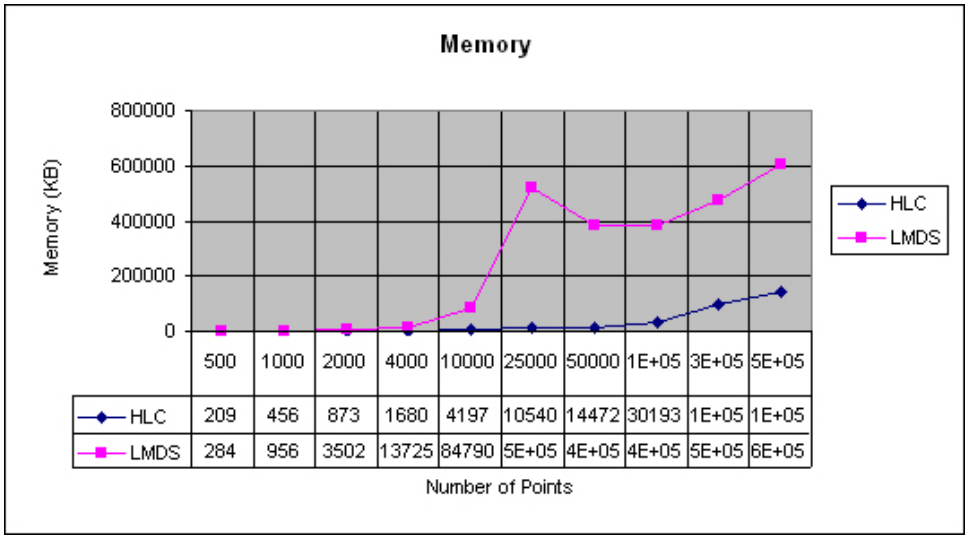


Figure 11: Memory Usages (KB) of LMDS vs HLC running on Swiss roll without noise.

6.2 Swiss roll with distortions

In this section, we considered the case that HLC performed better than Landmark MDS. In this section we applied Isomap, HLLE, LMDS, and HLC to distorted Swiss roll data sets ranging from 500 to 500,000 points. We found that LMDS was more sensitive to distortions and generated a slightly lower embedding quality than HLC since LMDS was dependent largely on landmark selection.

As illustrated in figure 12, we applied Isomap, HLLE, LMDS (with 10% randomly selected landmarks), and HLC (with chart size of 50, 50, 100, and 1000) to the distorted Swiss roll data sets consist of 500, 1000, 2000, and 4000 data points. It is apparent that HLLE gave us the best embedding quality while the others gave us the embeddings that contain noticeable deterioration.

Figure 13 shows the same data sets consist of 10,000, 25,000, 50,000, 100,000, 300,000, and 500,000 data points for which we applied LMDS (with 1000, 2500, 1000, 500, 200, and 150 randomly selected landmarks), and HLC (with chart size of 1000, 1000, 1000, 1000, 1000, and 500). Since the data sets were too large, we could not apply both Isomap and HLLE to these data sets due to the cost of the operations. The differences between the two algorithms are noticeable. The overall shape of the embedded data produced by HLC appears to be more rectangular than that of LMDS.

To illustrate that HLC gives us a better embedding quality than LMDS in this case, we considered a mean distance error of embedding data. Figure 14 shows the graphs that display a mean distance error produced by HLC and LMDS. The lower the mean distance error, the better the embedding quality, because the low value of a mean distance error means that there is a small difference between truth embedding and embedding produced by an algorithm. It was observed from the graph that mean distance errors of embedding data produced by HLC are significantly lower than those of LMDS.

In addition, residual variances of embedding data produced by HLC were lower than those of LMDS, as illustrated by figure 15, which means that the quality of embeddings produced by HLC in distorted data is better than that of LMDS.

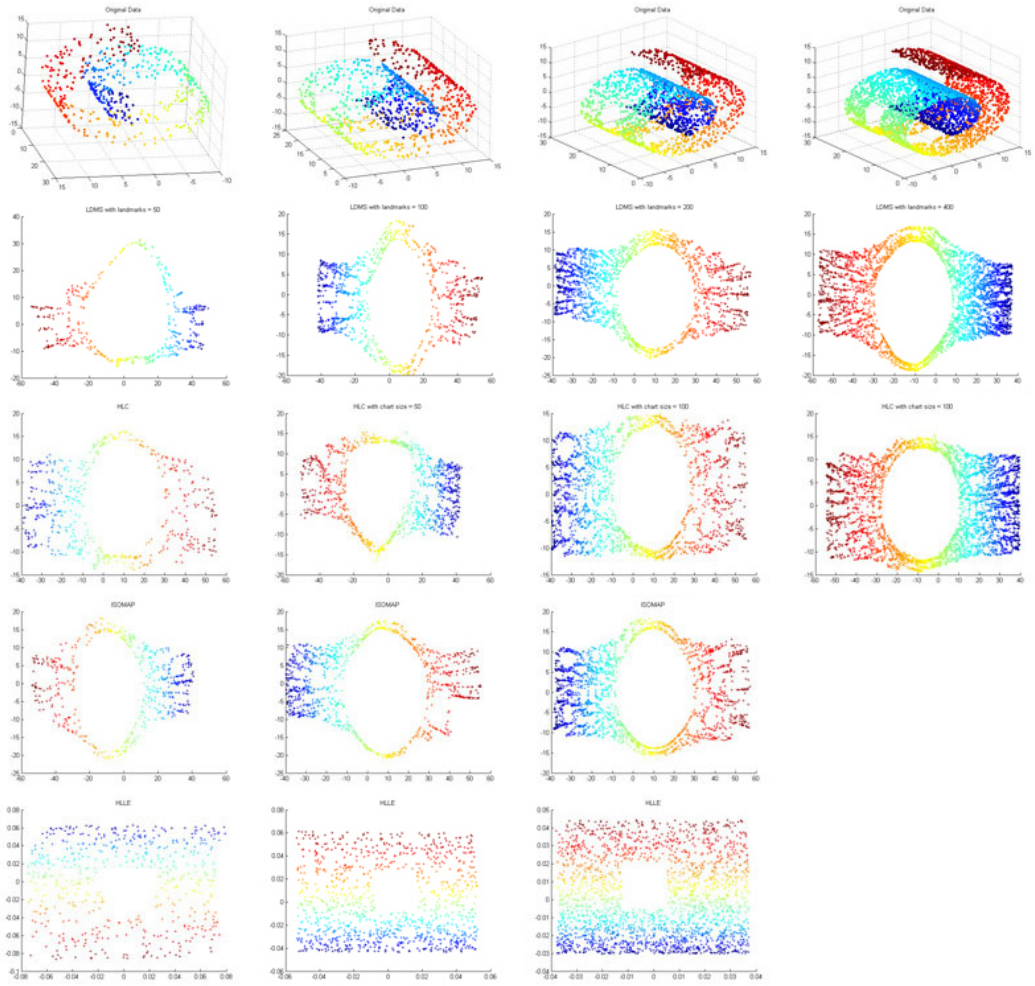


Figure 12: HLC is applied to a Swiss roll data set with distortions. The first row is the original data set of 500, 1000, 2000 and 4000 points. The second row is the embedding generated by LMDS with 10% landmark points. The third row is the embedding generated by HLC with chart size = 50, 50, 100 and 1000. The fourth row is the embedding generated by ISOMAP. The fifth row is the embedding generated by HLLE.

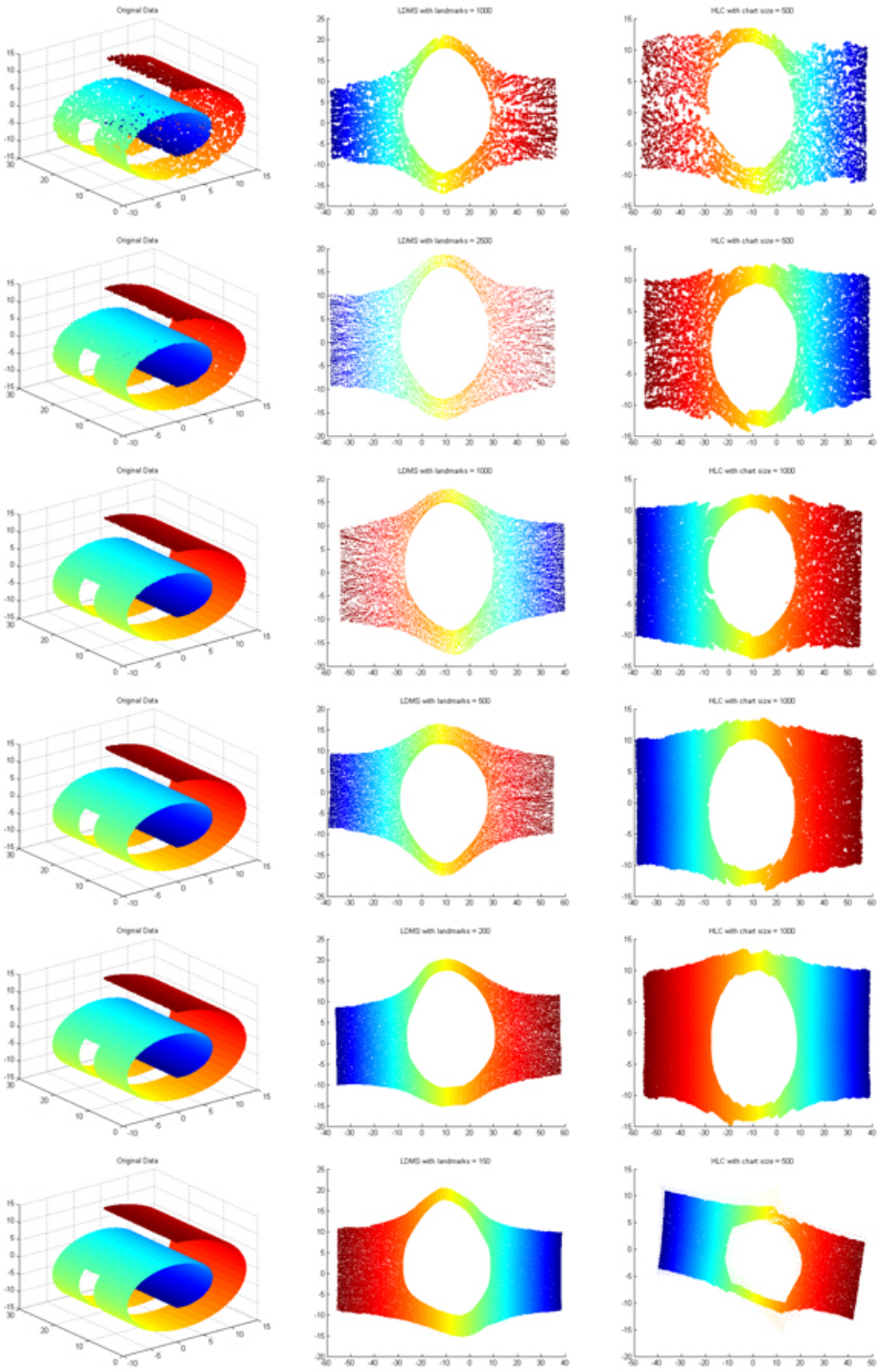


Figure 13: HLC is applied to a swiss roll data set with distortions. The first column is the original data set of 10k, 25k, 50k, 100k, 300k and 500k points. The second column is the embedding generated by LMDS with 1000, 2500, 1000, 500, 200 and 150 landmark points. The third column is the embedding generated by HLC with chart size = 1000 and 500.

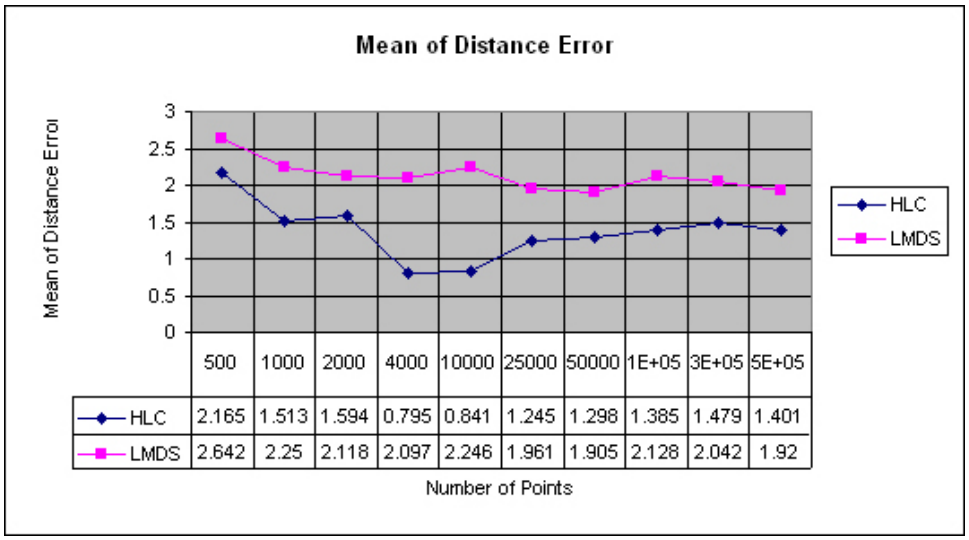


Figure 14: A Mean Distance Error of LMDS vs HLC running on Swiss roll with distortions.

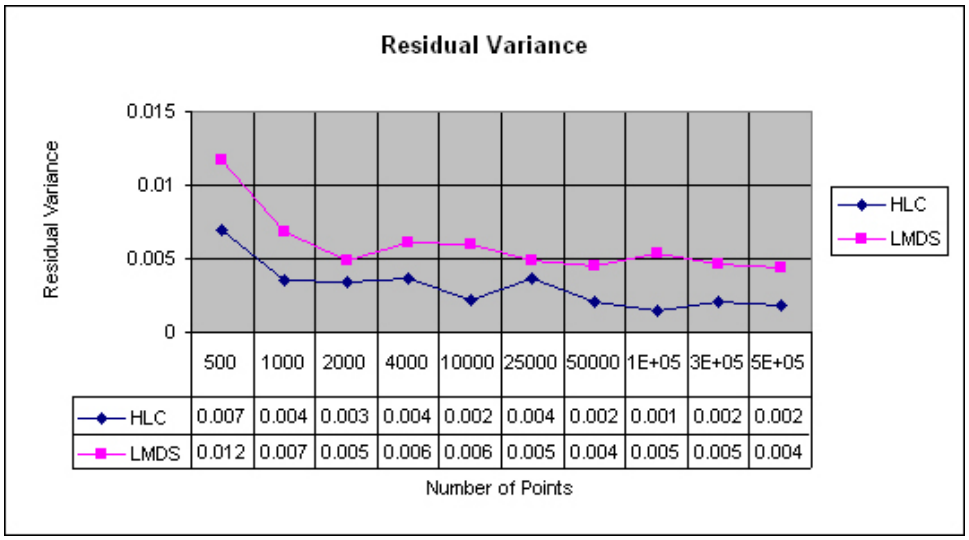


Figure 15: Residual Variances of LMDS vs HLC running on Swiss roll with distortions.

Figure 16 shows the graphs that display the running time (in seconds) required by HLC and LMDS. It can be observed from the graph that the running time increases as the number of input points increases, and the running times of both algorithms increase significantly when the number of input points is very large (100,000 to 500,000 points). The differences between the running times required by HLC and LMDS are significant when the number of input points is small. This is because the overhead occurred by means of the partition step, which does not have much effect on the large data sets. However, HLC requires a longer running time than LMDS.

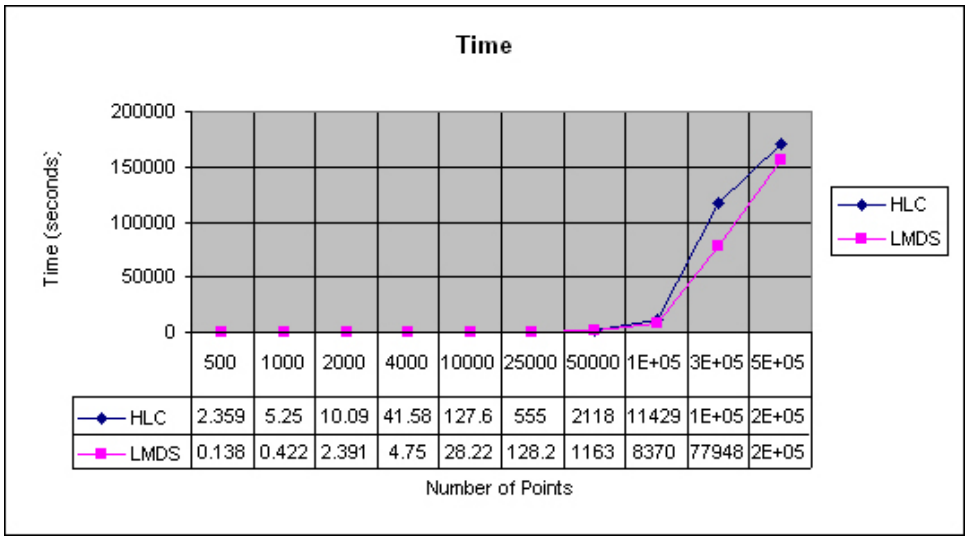


Figure 16: A running time (seconds) of LMDS vs HLC running on Swiss roll with distortions.

For the memory requirement, HLC needs a lower memory than LMDS, as illustrated in figure 17. The difference in memory requirements between HLC and LMDS is relatively miniscule when the data sets are small. However, when the data becomes large, the difference between the two algorithms becomes significant as well, owing to an expensive operation on an embedding step of LMDS. In contrast to LMDS, HLC keep data in storage and will be loaded into memory later as needed, which significantly influences the performance.

7 Conclusions

We have presented Hierarchical Landmark Charting as an extension to the Isomap algorithm to the non-linear dimensionality reduction problem when the number of data sets is large. Our method partitions data sets into a hierarchical structure, embeds each partition

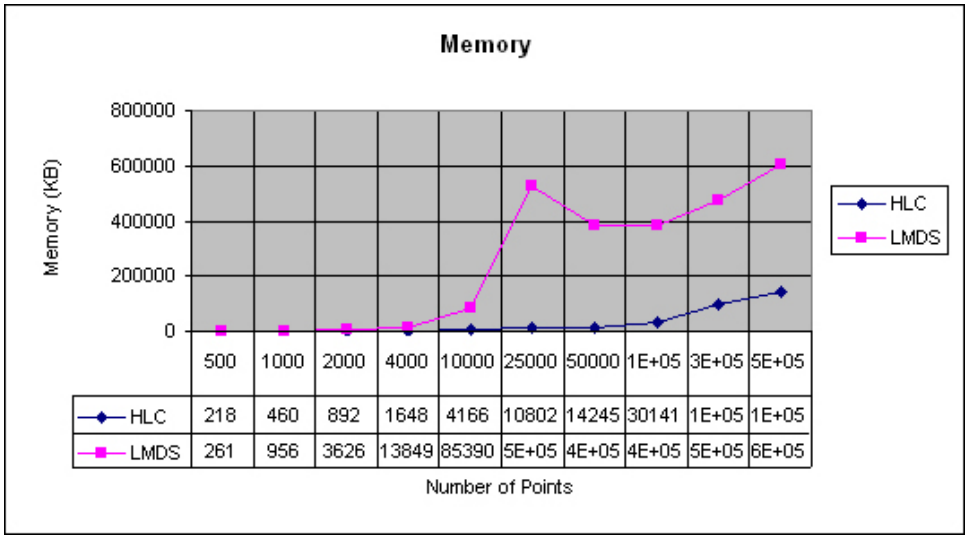


Figure 17: Memory Usages (KB) of LMDS vs HLC running on Swiss roll with distortions.

separately, and aligns them into single embedding using global landmark points. The main difference between HLC and LMDS is that LMDS embeds only landmark points and finds the coordinates of the remaining points using the geodesic distances, while HLC embeds each local neighborhood with global landmark points separately. We demonstrated HLC by running the charting on generated Swiss roll data sets ranging from 500 to 500,000 points and made a comparison to LMDS, Isomap, and HLLE. For a large volume of data sets, it is too expensive for Isomap and HLLE, so HLC and LMDS are more suitable for large size of data set. HLC and LMDS are considered equivalent since they gave the equivalent embedding quality in most cases. Unlike LMDS, HLC is not dependent largely on landmarks selection, so HLC generates better embedding quality in some cases, such as data that contains some distortions. HLC has an advantage over LMDS in term of the usage of memory since its memory requirements are independent of the number of data sets, and therefore, can run on larger size of data set, while there are not sufficient landmarks for running LMDS. However, HLC requires more running time than LMDS due to it needs to perform the embedding step, which is quite expensive, on every regions of the data sets. Depending on the application, one algorithm or the other may be most appropriate.

Acknowledgements

I would like to thank Daniel Grollman for modifying the HLC code which works much better than the old version.

References

[1] John C. Langford B. Tenenbaum, Vin de Silva. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[2] Vin de Silva and Joshua B. Tenenbaum. Local versus global methods for nonlinear dimensionality reduction. *Advances In Neural Information Processing System*, 15, 2003.

[3] Vin de Silva and Joshua B. Tenenbaum. Sparse multidimensional scaling using landmark points. *Stanford Mathematics Technical Report*, 2004.

[4] David L. Donoho and Carrie Grimes. Hessian eigenmaps: new locally linear embedding techniques for high-dimensional data. *Stanford Statistics Technical Report*, 2003–08.

[5] B. D. Packer K. Q. Weinberger and L. K. Saul. Nonlinear dimensionality reduction by semidefinite programming and kernel matrix factorization. *In Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AISTATS-05), Barbados*.

[6] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[7] Naonori Ueda Sean Stromsten Thomas L. Griffiths Tomoharu Iwata, Kazumi Saito and Joshua B. Tenenbaum. Parametric embedding for class visualization. *Advances in Neural Information Processing Systems*, 17:617–624, 2005.

[8] P. Vincent Y. Bengio, J-F. Paiement. Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. *Advances in Neural Information Processing System 16 (NIPS 2003)*, Dec 2003.

Appendix

Input:
$N \leftarrow$ number of data points;
$X \leftarrow$ data points;
$dimension \leftarrow$ low-dimensional resolution;
$g_x(i, j) \leftarrow$ global distances between data points i and j in input space X;
$k \leftarrow$ desired number of meta-neighborhood;
$b \leftarrow$ desired number of neighborhood;

Step 1: compute neighborhood matrix
for ($i = 1 : N$)
    $index \leftarrow$ sort($g_x(i, 1 : N)$); //get the sorted index
    for ($j = 1 : b$) //b nearest neighbors of point i
        $A(i, index(j)) \leftarrow g_x(i, index(j))$;

Step 2: Hierarchical partitions
$tree \leftarrow \emptyset$;
$level \leftarrow 1$;
$points \leftarrow X$;
$fpoints \leftarrow \emptyset$;
while (size($points$) $> k$)
    $chartnumber \leftarrow 1$;
    $free \leftarrow points$;
    while ($free \neq \emptyset$)
        $center \leftarrow$ random($free$); //pick a random point from free points
        $fpoints \leftarrow fpoints \cup center$;
        if ($level = 1$)
            $D_{center} \leftarrow$ dijkstra($A, center$);
            save D_{center} into storage;
        else
            load D_{center} from storage;
        $index \leftarrow$ sort(D_{center}); //get the index of nearest points to center
        $meta \leftarrow points(index(1 : k))$; //get the k nearest points
        $tree(level).chart(chartnumber).points \leftarrow meta$;
        $free \leftarrow free - meta$;
        $chartnumber \leftarrow chartnumber + 1$;
    $points \leftarrow fpoints$;
    $fpoints \leftarrow \emptyset$;
    $level \leftarrow level + 1$;

Figure 18: Pseudocode for HLC.

$global \leftarrow points$;
$tree(level).chart(1).points \leftarrow global$; //make the root node

Step 3: Embedding Charts
for ($level$ =size($tree$):1) //embed from root
    for ($chartnumber$ = 1:size($tree(level).chart$))
        $embedpoint \leftarrow tree(level).chart(chartnumber).points \cup global$;
        $embedmat \leftarrow A(embedpoint)$;
        if ($level$ = 1)
            $embedmat \leftarrow$ dijkstra($embedmat$,1:size($embedmat$));
        else
            $embedmat \leftarrow$ load distance for $embedpoint$ from storage;
        $coords \leftarrow$ Embed $embedmat$ using classical MDS;
        $tree(level).chart(chartnumber).coords \leftarrow coords$;

Step 4: Stitch together the charts
//use the global points' coordinates as reference
$Coords(1 : dimension, global) \leftarrow tree(length(tree)).chart(1).coords$;
$done \leftarrow global$;
for ($level = length(tree) - 1 : 1$)
    for ($chartnumber$ = 1:size($tree(level).chart$))
        $embedpoint \leftarrow tree(level).chart(chartnumber).points \cup global$;
        //find overlap points between all known points and this chart
        $overlap \leftarrow done \cap embedpoint$;
        //use the overlap points to find the transformation
        $LC \leftarrow tree(level).chart(chartnumber).coords(1 : dimension, overlap)$;
        $GC \leftarrow Coords(1 : dimension, overlap)$;
        $trans \leftarrow GC \times$ pinv(LC);
        $real \leftarrow trans \times tree(level).chart(chartnumber).coords$;
        $nonoverlap \leftarrow embedpoint - overlap$;
        //store the transformed locations.
        $Coords(1 : dimension, nonoverlap) \leftarrow real(1 : dimension, nonoverlap)$;
        $done = done \cup embedpoint$;

Output:
$Coords \rightarrow$ embedding data;

Figure 19: Pseudocode for HLC.