Visualization of Large High-Dimensional Data
via Interpolation Approach of Multidimensional Scaling

Seung-Hee Bae[a,*], Judy Qiu[a,b], Geoffrey Fox[a,b]

[a]*Pervasive Technology Institute at Indiana University, 2719 E. 10th St., Bloomington, IN, 47408, United States*
[b]*School of Informatics and Computing at Indiana University, 901 E. 10th St., Bloomington, IN, 47408, United States*

Abstract

The recent explosion of publicly available biology gene sequences and chemical compounds offers an unprecedented opportunity for data mining. To make data analysis feasible for such vast volume and high-dimensional scientific data, we can apply high performance dimension reduction algorithms. Among the known dimension reduction algorithms, we utilize the multidimensional scaling (MDS) algorithm to configure the given high-dimensional data into the target dimension. However, the MDS algorithm requires a quadratic order of memory as well as computing resources, and it is usually infeasible to deal with millions of points via normal MDS method even under a commodity distributed-memory cluster system. Thus, the authors propose a method of interpolation to utilizing the mapping of only a small subset of the given data. This approach effectively reduces computational complexity. With minor trade-off of approximation, interpolation method makes it possible to process millions of data points with modest amounts of computation and memory requirement. Since huge amount of data are dealt, we represent how to parallelize the proposed MDS interpolation algorithm, as well. For the evaluation of the interpolated MDS by STRESS criteria, it is necessary to compute symmetric all pairwise computation with only subset of required data per process, so we also propose a simple but efficient parallel mechanism for the symmetric all pairwise computation when only a subset of data is available to each process. Our experimental results illustrate that the quality of interpolated mapping results are comparable to the mapping results of original algorithm only. In parallel performance aspect, those interpolation methods are well parallelized with high efficiency. With the proposed MDS interpolation method, we construct a configuration of four-million *out-of-sample* data into the target dimension, and the number of *out-of-sample* data can be increased further.

Keywords: Multidimensional Scaling, Interpolation, Dimension Reduction, Data Visualization

1. Introduction

Due to the advancements in science and technology over the last several decades, every scientific and technical field has generated a huge amount of data as time has passed in the world. We are really in the era of data deluge. In reflecting on the data deluge era, data-intensive scientific computing [1] has emerged in the scientific computing fields and it has been attracting more by many people. To analyze those incredible amount of data, many data mining and machine learning algorithms have been developed. Among the many data mining and machine learning algorithms that have been invented, we focus on dimension reduction algorithms, which reduce data dimensionality from original high dimension to target dimension, in this paper.

Among the many dimension reduction algorithms which exist, such as principle component analysis (PCA), generative topographic mapping (GTM) [2, 3], self-organizing map (SOM) [4], multidimensional scaling (MDS) [5, 6], we have worked on MDS for this paper. Previously, we parallelize the MDS algorithm to utilize multicore clusters and to increase the

computational capability with minimal overhead for the purpose of investigating large data, such as 100000 (100k) data [7]. However, parallelization of an MDS algorithm, whose computational complexity and memory requirement is upto $O(N^2)$ where N is the number of points, is still limited by the memory requirement for huge data, e.g. millions of points, although it utilizes distributed memory environments, such as clusters, for acquiring more memory and computational resources. In this paper, we try to solve the memory-bound problem by applying an interpolation method based on pre-configured mappings of the sample data for the MDS algorithm, so that we can provide configuration of millions of points in the target space.

This paper is an extended version of the MDS interpolation part in [8], and this paper illustrates much more detailed experimental investigations and analyses of the proposed MDS interpolation. [1] This paper is organized as follows. First we will briefly discuss existing methods of *out-of-sample* problem in various dimension reduction algorithms and the concept of the multidimensional scaling (MDS) in Section 2 and Section 3, repectively. Then, the proposed interpolation method and how

---

*Corresponding author

Email addresses: `sebae@umail.iu.edu` (Seung-Hee Bae),
`xqiu@indiana.edu` (Judy Qiu), `gcf@indiana.edu` (Geoffrey Fox)

[1]An earlier version of this paper was presented at the 19th ACM International Simposium on High Performance Distributed Computing (HPDC 2010), and was published in its proceedings.

to parallelize it are described in Section 4. Various experimental analyses, which are related to the proposed MDS interpolation algorithm, are shown in Section 5, followed by our conclusion and future work in Section 6.

2. Related Work

The *out-of-sample* method, which embeds new points with respect to previously configured points, has been actively researched for recent years, and it aims at improving the capability of dimension reduction algorithms by reducing the computational and memory-wide requirement with the trade-off of slightly approximated mapping results.

In a sensor network localization field, when there are only a subset of pairwise distances between sensors and a subset of anchor locations are available, people try to find out the locations of the remaining sensors. For instance, the semi-definite programming relaxation approaches and its extended approaches has been proposed to solve this issue [9]. [10] and [11] proposed out-of-sample extension for the classical multidimensional scaling (CMDS) [12], which is based on spectral decomposition of a symmetric positive semidefinite matrix (or the approximation of positive semidefinite matrix), and the embeddings in the configured space are represented in terms of eigenvalues and eigenvectors of it. [10] projected the new point x onto the principal components, and [11] extends the CMDS algorithm itself to the out-of-sample problem. In [11], the authors described how to embed one point between the embeddings of the original n objects through modification of the original CMDS equations, which preserves the mappings of the original n objects, with $(n + 1) \times (n + 1)$ matrix A_2 instead of $n \times n$ matrix Δ_2, and extends to embedding a number of points simultaneously by using matrix operations. Recently, a multi-level force-based MDS algorithm was proposed as well [13].

In contrast to applying the out-of-sample problem to CMDS, I extend the out-of-sample problem to general MDS results with the STRESS criteria of Eq. (1), which finds embeddings of approximating to the distance (or dissimilarity) rather than the inner product as in CMDS, with an gradient descent optimization method, called iterative majorizing. The proposed iterative majorizing interpolation approach for the MDS problem will be explained in Section 4.

3. Multidimensional Scaling (MDS)

Multidimensional scaling (MDS) [12, 5, 6] is a general term that refers to techniques for constructing a map of generally high-dimensional data into a target dimension (typically a low dimension) with respect to the given pairwise proximity information. Mostly, MDS is used to visualize given high dimensional data or abstract data by generating a configuration of the given data which utilizes Euclidean low-dimensional space, i.e. two-dimension or three-dimension.

Generally, proximity information, which is represented as an $N \times N$ dissimilarity matrix ($\Delta = [\delta_{ij}]$), where N is the number of points (objects) and δ_{ij} is the dissimilarity between point i and

j, is given for the MDS problem, and the dissimilarity matrix (Δ) should agree with the following constraints: (1) symmetricity ($\delta_{ij} = \delta_{ji}$), (2) nonnegativity ($\delta_{ij} \geq 0$), and (3) zero diagonal elements ($\delta_{ii} = 0$). The objective of the MDS technique is to construct a configuration of a given high-dimensional data into low-dimensional Euclidean space, where each distance between a pair of points in the configuration is approximated to the corresponding dissimilarity value as much as possible. The output of MDS algorithms could be represented as an $N \times L$ configuration matrix X, whose rows represent each data point x_i ($i = 1, \ldots, N$) in L-dimensional space. It is quite straightforward to compute the Euclidean distance between x_i and x_j in the configuration matrix X, i.e. $d_{ij}(X) = \|x_i - x_j\|$, and we are able to evaluate how well the given points are configured in the L-dimensional space by using the suggested objective functions of MDS, called STRESS [14] or SSTRESS [15]. Definitions of STRESS (1) and SSTRESS (2) are following:

$$\sigma(X) \quad = \quad \sum_{i<j\leq N} w_{ij}(d_{ij}(X) - \delta_{ij})^2 \qquad (1)$$

$$\sigma^2(X) \quad = \quad \sum_{i<j\leq N} w_{ij}[(d_{ij}(X))^2 - (\delta_{ij})^2]^2 \qquad (2)$$

where $1 \leq i < j \leq N$ and w_{ij} is a weight value, so $w_{ij} \geq 0$.

As shown in the STRESS and SSTRESS functions, the MDS problems could be considered to be non-linear optimization problems, which minimizes the STRESS or the SSTRESS function in the process of configuring an L-dimensional mapping of the high-dimensional data.

4. Majorizing Interpolation MDS

One of the main limitation of most MDS applications is that they require $O(N^2)$ memory as well as $O(N^2)$ computation. Thus, though it is possible to run them with a small data size without any trouble, it is impossible to execute them with a large number of points due to memory limitation; therefore, this challenge could be considered as being a memory-bound problem as well as a computing-bound problem. For instance, Scaling by MAjorizing of COmplicated Function (SMACOF) [16, 17], a well-known MDS application via Expectation-Maximization (EM) [18] approach, uses six $N \times N$ matrices. If $N = 100,000$, then one $N \times N$ matrix of 8-byte double-precision numbers requires 80 GB of main memory, so the algorithm needs to acquire at least 480 GB of memory to store these six $N \times N$ matrices. It is possible to run a parallel version of SMACOF with MPI in **Cluster-II** in Table 1 with $N = 100,000$. If the data size is increased only twice, however, then the SMACOF algorithm should have 1.92 TB of memory, which is bigger than the total memory of **Cluster-II** in Table 1 (1.536 TB), so it is impossible to run it effectively within the cluster. Increasing memory size will not be a solution, even though it could increase the runnable number of points. It will encounter the same problem as the data size increases.

To solve this obstacle, we develop a simple interpolation approach based on pre-mapped MDS result of the sample of the

2

given data. Our interpolation algorithm is similar to the k nearest neighbor (k-NN) classification [19], but we approximate a new mapping position of the new point based on the positions of k-NN, among pre-mapped subset data, instead of classifying it. For the purpose of deciding a new mapping position in relation to the k-NN positions, the iterative majorization method is applied as in the SMACOF [16, 17] algorithm. The details of mathematical majorization equations for the proposed out-of-sample MDS algorithm is shown below. The algorithm proposed in this paper is called Majorizing Interpolation MDS (hereafter *MI-MDS*).

The proposed algorithm is implemented as follows. We are given N data in a high-dimensional space, say D-dimension, and proximity information ($\Delta = [\delta_{ij}]$) of those data as in Section 3. Among N data, the configuration of the n sample points in L-dimensional space, $x_1, \ldots, x_n \in \mathbb{R}^L$, called X, are already constructed by an MDS algorithm; here we use the SMACOF algorithm. Then, we select k nearest neighbors ($p_1, \ldots, p_k \in P$) of the given new point, among n pre-mapped points with respect to corresponding δ_{ix}, where x represents the new point. I use a linear search to find the k-nearest neighbors among n-sampled data, so that the complexity of finding the k-nearest neighbors is $O(n)$ per one interpolated point (here x). Finally, the new mapping of the given new point $x \in \mathbb{R}^L$ is calculated based on the pre-mapped position of the selected k-NN and the corresponding proximity information δ_{ix}. The finding new mapping position is considered as a minimization problem of STRESS (3) as similar as normal MDS problem with m points, where $m = k+1$. However, only one point x is movable among m points, so we can simplify the STRESS equation (3) as follows (Eq. (4)), and we set $w_{ij} = 1$, for $\forall i, j$ in order to simplify.

$$\sigma(X) = \sum_{i<j\leq m} (d_{ij}(X) - \delta_{ij})^2 \tag{3}$$

$$= C + \sum_{i=1}^{k} d_{ix}^2 - 2\sum_{i=1}^{k} \delta_{ix}d_{ix} \tag{4}$$

where δ_{ix} is the original dissimilarity value between p_i and x, d_{ix} is the Euclidean distance in L-dimension between p_i and x, and C represents a constant part. The second term of Eq. (4) can be deployed as following:

$$\sum_{i=1}^{k} d_{ix}^2 = \|x - p_1\|^2 + \cdots + \|x - p_k\|^2 \tag{5}$$

$$= k\|x\|^2 + \sum_{i=1}^{k} \|p_i\|^2 - 2x^t q \tag{6}$$

where $q^t = (\sum_{i=1}^{k} p_{i1}, \ldots, \sum_{i=1}^{k} p_{iL})$ and p_{ij} represents j-th element of p_i. In order to establish majorizing inequality, we apply *Cauchy-Schwarz* inequality to $-d_{ix}$ of the third term of Eq. (4). Please, refer to chapter 8 in [6] for details of how to apply the *Cauchy-Schwarz* inequality to $-d_{ix}$. Since $d_{ix} = \|p_i - x\|$, $-d_{ix}$ could have following inequality based on *Cauchy-Schwarz* inequality:

$$-d_{ix} \leq \frac{\sum_{a=1}^{L}(p_{ia} - x_a)(p_{ia} - z_a)}{d_{iz}} \tag{7}$$

$$= \frac{(p_i - x)^t(p_i - z)}{d_{iz}} \tag{8}$$

where $z^t = (z_i, \ldots, z_L)$ and $d_{iz} = \|p_i - z\|$. The equality in Eq. (7) occurs if x and z are equal. If we apply Eq. (8) to the third term of Eq. (4), then we obtain

$$-\sum_{i=1}^{k} \delta_{ix}d_{ix} \leq -\sum_{i=1}^{k} \frac{\delta_{ix}}{d_{iz}}(p_i - x)^t(p_i - z) \tag{9}$$

$$= -x^t \sum_{i=1}^{k} \frac{\delta_{ix}}{d_{iz}}(z - p_i) + C_\rho \tag{10}$$

where C_ρ is a constant. If Eq. (6) and Eq. (10) are applied to Eq. (4), then it could be like following:

$$\sigma(X) = C + \sum_{i=1}^{k} d_{ix}^2 - 2\sum_{i=1}^{k} \delta_{ix}d_{ix} \tag{11}$$

$$\leq C + k\|x\|^2 - 2x^t q + \sum_{i=1}^{k} \|p_i\|^2$$

$$- 2x^t \sum_{i=1}^{k} \frac{\delta_{ix}}{d_{iz}}(z - p_i) + C_\rho \tag{12}$$

$$= \tau(x, z) \tag{13}$$

where both C and C_ρ are constants. In the Eq. (13), $\tau(x, z)$, a quadratic function of x, is a majorization function of the STRESS. Through setting the derivative of $\tau(x, z)$ equal to zero, we can obtain a minimum of it; that is

$$\nabla\tau(x, z) = 2kx - 2q - 2\sum_{i=1}^{k} \frac{\delta_{ix}}{d_{iz}}(z - p_i) = 0 \tag{14}$$

$$x = \frac{q + \sum_{i=1}^{k} \frac{\delta_{ix}}{d_{iz}}(z - p_i)}{k} \tag{15}$$

where $q^t = (\sum_{i=1}^{k} p_{i1}, \ldots, \sum_{i=1}^{k} p_{iL})$, p_{ij} represents j-th element of p_i, and k is the number of the nearest neighbors that we selected.

The advantage of the iterative majorization algorithm is that it produces a series of mappings with non-increasing STRESS values as proceeds, which results in local minima. It is good enough to find local minima, since the proposed MI-MDS algorithm simplifies the complicated non-linear optimization problem as a small non-linear optimization problem, such as $k + 1$ points non-linear optimization problem, where $k \ll N$. Finally, if we substitute z with $x^{[t-1]}$ in Eq. (15), then we generate an iterative majorizing equation like the following:

$$x^{[t]} = \frac{q + \sum_{i=1}^{k} \frac{\delta_{ix}}{d_{iz}}(x^{[t-1]} - p_i)}{k} \qquad (16)$$

$$x^{[t]} = \overline{p} + \frac{1}{k} \sum_{i=1}^{k} \frac{\delta_{ix}}{d_{iz}}(x^{[t-1]} - p_i) \qquad (17)$$

where $d_{iz} = \|p_i - x^{[t-1]}\|$ and $\overline{p}$ is the average of k-NN's mapping results. Eq. (17) is an iterative equation used to embed newly added point into target-dimensional space, based on pre-mapped positions of k-NN. The iteration stop condition is essentially the same as that of the SMACOF algorithm, which is

$$\Delta\sigma(S^{[t]}) = \sigma(S^{[t-1]}) - \sigma(S^{[t]}) < \varepsilon, \qquad (18)$$

where $S = P \cup \{x\}$ and ε is the given threshold value.

The time complexity of the proposed MI-MDS algorithm to find the mapping of one interpolated point is $O(k)$ on the basis of Eq. (17), if we assume that the number of iterations of finding one interpolated mapping is very small. Since finding nearest neighbors takes $O(n)$ and mapping via MI-MDS requires $O(k)$ for one interpolated point, the overall time complexity to find mappings of overall out-of-sample points (N-n points) via the proposed MI-MDS algorithm is $O(kn(N - n)) \approx O(n(N - n))$, due to the fact that k is usually negligible compared to n or N.

The process of the overall out-of-sample MDS with a large dataset could be summarized by the following steps: (1) Sampling; (2) Running MDS with sample data; and (3) Interpolating the remain data points based on the mapping results of the sample data.

Alg. 1 describes the summary of the proposed MI-MDS algorithm for interpolation of a new data, say x, in relation to pre-mapping result of the sample data. Note that the algorithm uses $\overline{p}$ as an initial mapping of the new point $x^{[0]}$ unless initialization with $\overline{p}$ makes $d_{ix} = 0$, since the mapping is based on the k-NN. $\overline{p}$ makes $d_{ix} = 0$, if and only if all the mapping positions of the k-NNs are on the same position. If $\overline{p}$ makes $d_{ix} = 0$ ($i = 1, \ldots, k$), then we generate a random variation from the $\overline{p}$ point with the average distance of δ_{ix} as an initial position of $x^{[0]}$.

4.1. Parallel MI-MDS Algorithm

Suppose that, among N points, the mapping results of n sample points in the target dimension, say L-dimension, are given so that we could use those pre-mapped results of n points via MI-MDS algorithm which is described above to embed the remaining points ($M = N - n$). Though interpolation approach is much faster than full running MDS algorithm, i.e. $O(Mn + n^2)$ vs. $O(N^2)$, implementing parallel MI-MDS algorithm is essential, since M can be still huge, like millions. In addition, most of clusters are now in forms of multicore-clusters after the invention of the multicore-chip, so we are using hybrid-model parallelism, which combine processes and threads together as used in [20, 1].

In contrast to the original MDS algorithm in which the mapping of a point is influenced by the other points, interpolated points are totally independent one another, except selected k-NN in the MI-MDS algorithm, and the independency

Algorithm 1 Majorizing Interpolation MDS (MI-MDS) algorithm

1: Find k-NN: find k nearest neighbors of x, $p_i \in P$ $i = 1, \ldots, k$ of the given new data based on original dissimilarity δ_{ix}.
2: Gather mapping results in target dimension of the k-NN.
3: Calculate $\overline{p}$, the average of pre-mapped results of $p_i \in P$.
4: Generate initial mapping of x, called $x^{[0]}$, either $\overline{p}$ or a random variation from $\overline{p}$ point.
5: Compute $\sigma(S^{[0]})$, where $S^{[0]} = P \cup \{x^{[0]}\}$.
6: **while** $t = 0$ or ($\Delta\sigma(S^{[t]}) > \varepsilon$ and $t \leq$ MAX_ITER) **do**
7:    increase t by one.
8:    Compute $x^{[t]}$ by Eq. (17).
9:    Compute $\sigma(S^{[t]})$.
10: **end while**
11: **return** $x^{[t]}$;

of among interpolated points makes the MI-MDS algorithm to be pleasingly-parallel. In other words, there must be minimum communication overhead. Also, load-balance can be achieved by using modular calculation to assign interpolated points to each parallel unit, either between processes or between threads, as the number of assigned points are different at most one.

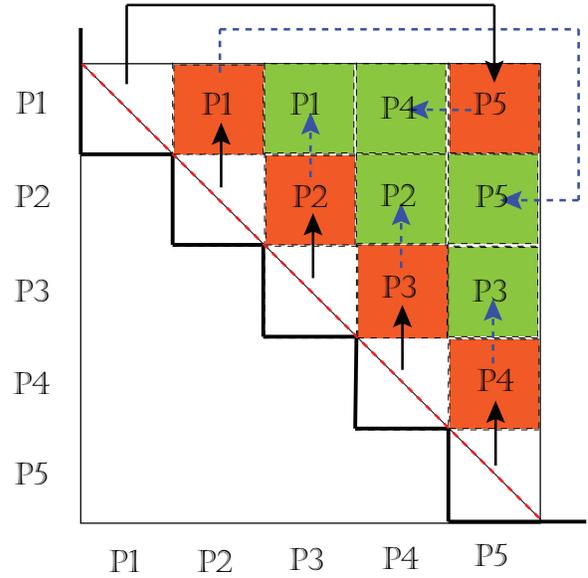### 4.2. Parallel Pairwise Computation Method with Subset of Data

Figure 1: Message passing pattern and parallel symmetric pairwise computation for calculating STRESS value of whole mapping results.

Although interpolation approach itself is in $O(Mn)$, if we want to evaluate the quality of the interpolated results by STRESS criteria of Eq. (1) of overall N points, it requires $O(N^2)$ computation. Note that we implement our hybrid-parallel MI-MDS algorithm as each process has access to only a subset of M interpolated points, without loss of generality M/p points, as

4

well as the information of all pre-mapped n points. It is natural way of using a distributed-memory system, such as cluster systems, to access only a subset of huge data which spread to over the clusters, so that each process needs to communicate each other for the purpose of accessing all the necessary data to compute STRESS.

In this section, we illustrate how to calculate symmetric pairwise computation efficiently in parallel with the case that only a subset of data is available for each process. In fact, general MDS algorithms utilize pairwise dissimilarity information, but suppose we are given N original vectors in D-dimension, $y_1, \ldots, y_N \in Y$ and $y_i \in \mathbb{R}^D$, instead of a given dissimilarity matrix, as PubChem finger print data that we used for our experiments. Thus, in order to calculate the distance in original D-dimension $\delta_{ij} = \|y_i - y_j\|$ in Eq. (1), it is necessary to communicate messages between each process to get the required original vector, say y_i and y_j. Here, we used the proposed pairwise computation method to measure the STRESS criteria of MDS problem in Eq. (1), but the proposed parallel pairwise computation method will be used efficiently for general parallel pairwise computation whose computing components are independent, such as generating a distance (or dissimilarity) matrix of all data, under the condition that each process can access only a subset of the required data.

Fig. 1 describes the proposed scheme when the number of processes (p) is 5, odd numbers. The proposed scheme is an iterative two-step approach, (1) rolling and (2) computing, and the iteration number is $\lceil (1 + \cdots + p - 1)/p \rceil = \lceil (p - 1)/2 \rceil$. Note that iteration ZERO is calculating the upper triangular part of the corresponding diagonal block, which does not requires message passing. After iteration ZERO is done, each process p_i sends the originally assigned data block to the previous process p_{i-1} and receives a data block from the next process p_{i+1} in cyclic way. For instance, process p_0 sends its own block to process p_{p-1}, and receives a block from process p_1. This rolling message passing can be done using one single MPI primitive per process, MPI_SENDRECV(), which is efficient. After sending and receiving messages, each process performs currently available pairwise computing block with respect to receiving data and originally assigned block. In Fig. 1, black solid arrows represent each message passings at iteration 1, and orange blocks with process ID are the calculated blocks by the corresponding named process at iteration 1. From iteration 2 to iteration $\lceil (p - 1)/2 \rceil$, the above two-steps are done repeatedly and the only difference is nothing but sending a received data block instead of the originally assigned data block. The green blocks and dotted blue arrows show iteration 2 which is the last iteration for the case of $p = 5$.

Also, for the case that the number of processes is even, the above two-step scheme works in high efficiency. The only difference between the odd number case and the even number case is that two processes are assigned to one block at the last iteration of even number case, but not in an odd number case. Though two processes are assigned to a single block, it is easy to achieve load balance by dividing two sections of the block and assigning them to each process. Therefore, both odd number process and even number process cases are parallelized well

---

Algorithm 2 Parallel Pairwise Computation

1: **input:** Y = a subset of data;
2: **input:** p = the number of process;
3: $rank \Leftarrow$ the rank of process;
4: $sendTo \Leftarrow (rank - 1) \bmod p$
5: $recvFrom \Leftarrow (rank + 1) \bmod p$
6: $k \Leftarrow 0$;
7: Compute upper triangle in the diagonal blocks in Fig. 1;
8: $MAX\_ITER \Leftarrow \lceil (p - 1)/2 \rceil$

9: **while** $k < MAX\_ITER$ **do**
10:     $k \Leftarrow k + 1$;
11:     **if** $k = 1$ **then**
12:         MPI_SENDRECV(Y, $sendTo$, Y_r, $recvFrom$);
13:     **else**
14:         $Y_s \Leftarrow Y_r$;
15:         MPI_SENDRECV(Y_s, $sendTo$, Y_r, $recvFrom$);
16:     **end if**

17:     Do Computation();
18: **end while**

---

using the above rolling-computing scheme, with minimal message passing overhead. The summary of the above parallel pairwise computation is shown in Alg. 2.

5. Analysis of Experimental Results

To measure the quality and parallel performance of the proposed MDS interpolation (*MI-MDS*) approach discussed in this paper, we have used 166-dimensional chemical dataset obtained from the PubChem project database[2], which is an NIH-funded repository for over 60 million chemical molecules and provides their chemical structures and biological activities, for the purpose of chemical information mining and exploration. In this paper we have used observations which consist of randomly selected up to 4 million chemical subsets for our testing. The computing cluster systems we have used in our experiments are summarized in Table 1.

In the following, we will mainly show: i) exploration of the optimal number of nearest neighbors; ii) the quality of the proposed MI-MDS interpolation approach in performing MDS algorithms, with respect to various sample sizes – 12500 (12.5k), 25000 (25k), and 50000 (50k) randomly selected from 100000 (100k) dataset as a basis – as well as the mapping results of large-scale data, i.e. up to 4 million points; and iii) parallel performance measurements of our parallelized interpolation algorithms on our clustering systems as listed in Table 1; and finally, iv) our results on processing up to 4 million MDS maps based on the trained result from 100k dataset.

5.1. Exploration of optimal number of nearest neighbors

Generally, the quality of k-NN (k-nearest neighbor) classification (or regression) is related to the number of neighbors. For

---

Features	Cluster-I	Cluster-II
# Nodes	8	32
CPU	AMD Opteron 8356 2.3GHz	Intel Xeon E7450 2.4 GHz
# CPU / # Cores per node	4 / 16	4 / 24
Total Cores	128	768
Memory per node	16 GB	48 GB
Network	Giga bit Ethernet	20 Gbps Infiniband
Operating System	Windows Server 2008 HPC Edition (Service Pack 2) - 64 bit	Windows Server 2008 HPC Edition (Service Pack 2) - 64 bit
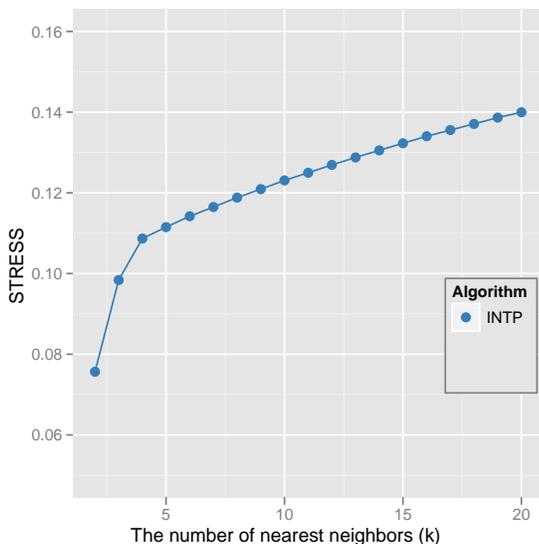
Figure 2: Quality comparison between interpolated result of 100k with respect to the number of nearest neighbors (k) with 50k sample and 50k out-of-sample result.

instance, if we choose a larger number for the k, then the algorithm shows a higher bias but lower variance. On the other hands, the k-NN algorithm shows a lower bias but a higher variance based on a smaller number of neighbors. For the case of k-NN classification, the optimal number of nearest neighbors (k) can be determined by the *N-fold cross validation method* [21] or *leave-one-out cross validation method*, and usually a k value that minimizes the cross validation error is picked.

Although we cannot use the N-fold cross validation method to decide the optimal k value of the proposed MI-MDS algorithm, we can compare the mapping results with respect to k value based on STRESS value. In order to explore the optimal number of nearest neighbors, we experimented with the MI-MDS algorithm with different k values, i.e. $2 \leq k \leq 20$ with 100k pubchem data.

Fig. 2 shows the comparison of mapping quality between the MI-MDS results of 100k data with 50k sample data size in terms of different k values. The y-axis of the plot is the *normalized STRESS* value which is divided by $\sum_{i<j} \delta_{ij}^2$. The normal-

ized STRESS value is equal to ONE when all the mapping is at the same position, in that the normalized STRESS value denotes the relative portion of the squared distance error rates of the given data set without regard to various scales of δ_{ij} due to data difference. The equation of normalized STRESS is shown in Eq. (19) below.

$$\sigma(\boldsymbol{X}) = \sum_{i<j\leq N} \frac{1}{\sum_{i<j} \delta_{ij}^2} (d_{ij}(\boldsymbol{X}) - \delta_{ij})^2 \qquad (19)$$

Fig. 2 shows an interesting result that the optimal number of nearest neighbors is 'two' rather than larger values. Also, the normalized STRESS value is statically increased as k is increased, when $k = 5$ and larger, and the normalized STRESS value of MI-MDS results with 20-NN is almost double of that with 2-NN.

Before we analyze the optimal number of nearest neighbors for the proposed MI-MDS method with the given Pubchem dataset, I would like to mention how the proposed MI-MDS solves the mapping ambiguity problem when $k = 2, 3$ for three dimensional target space. When the target dimension is 3D space, logically, the optimal position of the interpolated points can be in a circle if $k = 2$, and the optimal position of the interpolated points can be at two symmetric positions with respect to the face contains all three nearest neighbors, in the case of $k = 3$. The derivative MI-MDS equation in Eq. (17), however, constrains the interpolation space corresponding to the number of nearest neighbors, by setting the initial position as the average of the mappings of nearest neighbors. In the case of $k = 2$, the interpolation space is constructed as a line ($\boldsymbol{l}$) which includes the mapping positions of the two nearest neighbors, when the initial mapping of the interpolation is the center of nearest neighbors ($\overline{\boldsymbol{p}}$). Similarly, the possible mapping position of the interpolated point is constrained within the Face ($\boldsymbol{F}$) when it contains the three nearest neighbors when $k = 3$.

Fig. 3-(a) and (b) illustrate the constrained interpolation space in case of $k = 2$ and 3, correspondingly. In Fig. 3, $\boldsymbol{x_0}$ represents the initial mapping position of the interpolated point which is the same as $\overline{\boldsymbol{p}}$ and $\boldsymbol{v_i}$ ($i = 1, 2$ or 3) is the vector representation of $\boldsymbol{x_c} - \boldsymbol{p_i}$, where $\boldsymbol{x_c}$ is the current mapping position of the interpolated point and $\boldsymbol{p_i}$ is the mapping position in target dimension of nearest neighbors. Note that $\boldsymbol{x_0}$ ($= \overline{\boldsymbol{p}}$) is on the line l when $k = 2$ and on the face F when $k = 3$. If $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$
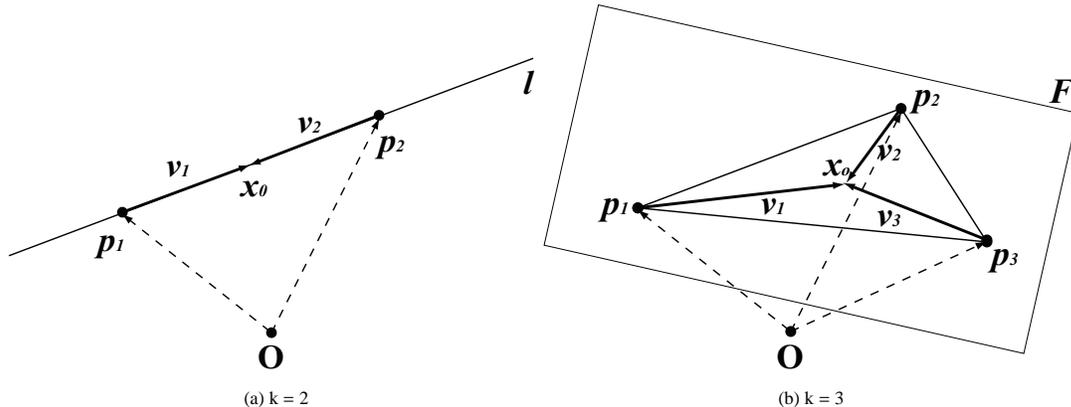
Figure 3: The illustration of the constrained interpolation space when $k = 2$ or $k = 3$ by initialization at the center of the mappings of the nearest neighbors.

are on the same line l, $\alpha v_1 + \beta v_2$ is also on the same line l. Similarly, if v_1, v_2, and v_3 are on the same Face F, $\alpha v_1 + \beta v_2 + \gamma v_3$ is also on the same face F. Thus, the final mapping position of the interpolated point with $k = 2$ or 3 is constrained in the line l or face F, as shown in Fig. 3. This results in removing the ambiguity of the optimal mapping position of the small nearest neighbor cases, for example $k = 2, 3$ when the target dimension is 3.

We can think of two MI-MDS specific properties as possible reasons for the results of the experiment of the optimal number of nearest neighbors which is shown in Fig. 2. A distinct feature of MI-MDS algorithm compared to other k-NN approaches is that the increase of the number of nearest neighbors results in generating more a complicated problem space to find the mapping position of the newly interpolated point. Note that the interpolation approach allows only the interpolated point to be moved and the selected nearest neighbors are fixed in the target dimension. This algorithmic property effects more severe constraints to find optimal mapping position with respect to Eq. (4). Also, note that finding the optimal interpolated position does not guarantee that it makes better mapping in terms of full data mapping, but it means that MI-MDS algorithm works as the algorithm designed.

Another specific property of MI-MDS is that the purpose of the MI-MDS algorithm is to find appropriate embeddings for the new points based on the given mappings of the sample data. Thus, it could be better to be sensitive to the mappings of closely-located nearest neighbors of the new point than to be biased to the distribution of the mappings of whole sample points. Fig. 4 illustrates the mapping difference with respect to the number of nearest neighbors used for MI-MDS algorithm with 50k sample and 50k out-of-sample data. The 50k sample data is selected randomly from the given 100k data set, so it is reasonable that the sampled 50k data and out-of-sample 50k show similar distributions. As shown in Fig. 4-(a), the interpolated points are distributed similar to the sampled data as we expected. Also, Fig. 4-(a) are much more similar to the configuration of the full MDS running with 100k data, which is shown later in this paper, than other results in Fig. 4. On the other hand, Fig. 4-(b) through Fig. 4-(f) are shown in center-biased

mapping, and the degree of bias of those mappings increases as k increases.

In order to understand more about why biased mappings are generated by larger nearest neighbors cases with the test dataset, we have investigated the given original distance distribution of the 50k sampled data set and the trained mapping distance distribution of the sampled data. Also, we have analyzed the training mapping distance between k-NNs with respect to k. Fig. 5 is the histogram of the original distance distribution and the trained mapping distance distribution of 50k sampled data used in Fig. 2 and Fig. 4. As shown in Fig. 5, most of the original distances are in between 5 and 7, but the trained mapping distances reside in a more broad interval. Table 2 demonstrates the distribution of the maximum mapping distance between selected k-NNs with respect to the number of nearest neighbors. The maximum original distance is 10.198 and the maximum mapping distance of the 50k sampled data is 12.960.

As shown in Fig. 16-(a), the mapping of Pubchem data forms a spherical shape. Thus, the maximum mapping distance of the 50k sampled data could be similar to the diameter of the spherical mapping. The distance 3.0 is close to the half of radius of the sphere and the distance 6.0 is close to the radius of the sphere. Therefore, in Table 2, the column of "< **3.0**" represents the cases that nearest neighbors are closely mapped together, and the columns of "> **6.0**" and others illustrate the cases that some nearest neighbors are far from other nearest neighbors. Note that the entries of "> **6.0**" column include that of "> **7.0**" and "> **8.0**" as well.

The analysis of mapping distance between k-NNs with the tested Pubchem dataset shows interesting results. Initially, we expected that $k = 5$ or $k = 10$ could be small enough numbers of the nearest neighbors, which would make nearest neighbors be positioned near each other in the training mapping results. Contrary to our expectation, as shown in Table 2, even in the case of $k = 2$, nearest neighbors are not near each other for some interpolated data. The cases of two nearest neighbors positioned more than a 6.0 distance occurred more than 400 times. As we increase k to be equal to 3, the occurrence of the cases of at least two nearest neighbors distanced more than 6.0 increases more than twice of what it was when $k = 2$. On the other hand,

7

(a) k = 2

(b) k = 3

(c) k = 5

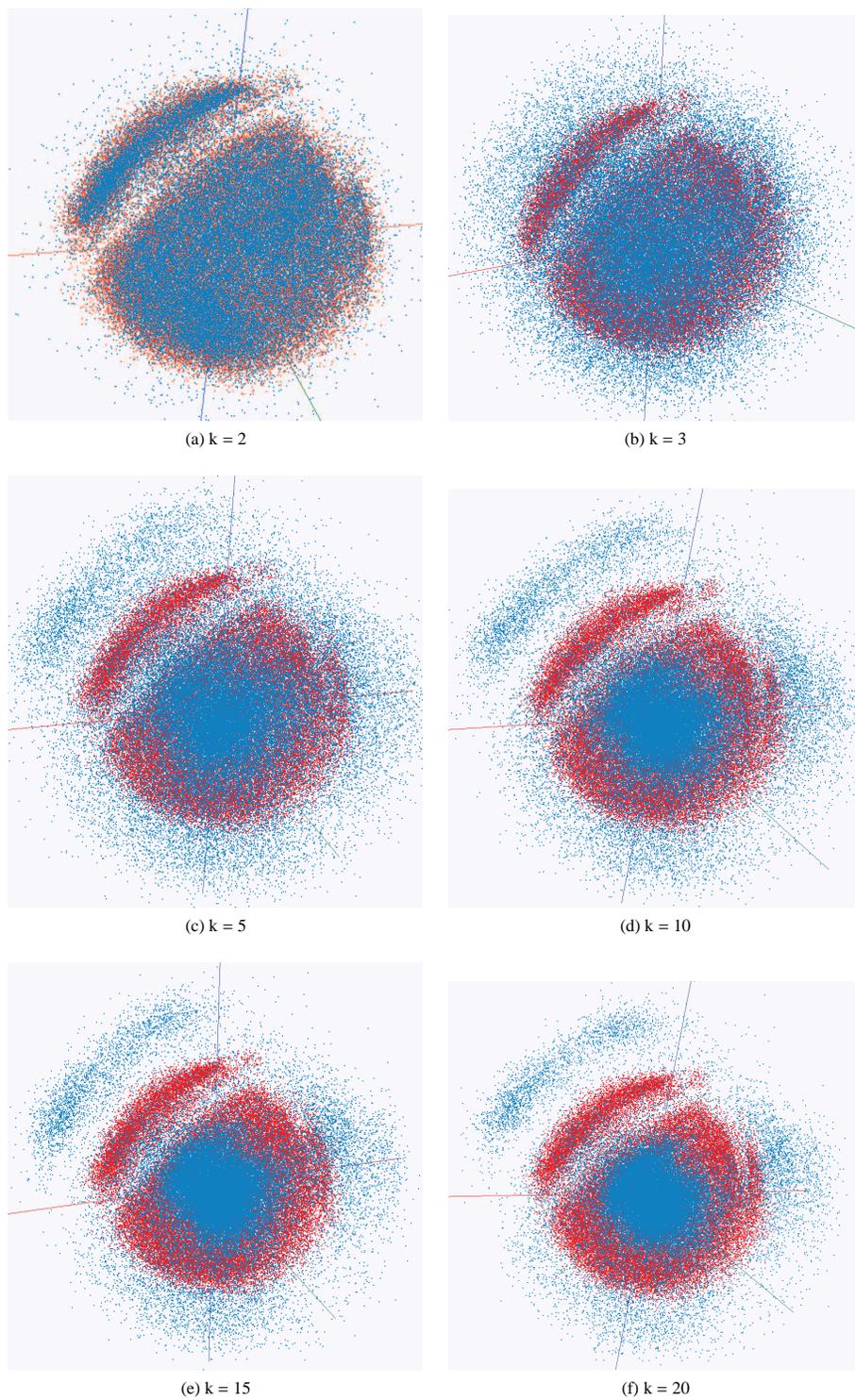(d) k = 10

(e) k = 15

(f) k = 20

Figure 4: The mapping results of MI-MDS of 100k Pubchem data with 50k sample data and 50k out-of-sample data with respect to the number of nearest neighbors (*k*). The sample points are shown in red and the interpolated points are shown in blue.

Table 2: Analysis of Maximum Mapping Distance between k-NNs with respect to the number of nearest neighbors (k).

#k-NNs	< 3.0	> 6.0	> 7.0	> 8.0	% of (< 3.0)	% of (> 6.0)
2	45890	409	164	53	91.780	0.818
3	41772	916	387	139	83.544	1.832
5	34503	1945	867	334	69.006	3.890
10	22004	4230	2005	826	44.008	8.460
20	10304	8134	4124	1797	20.608	16.268
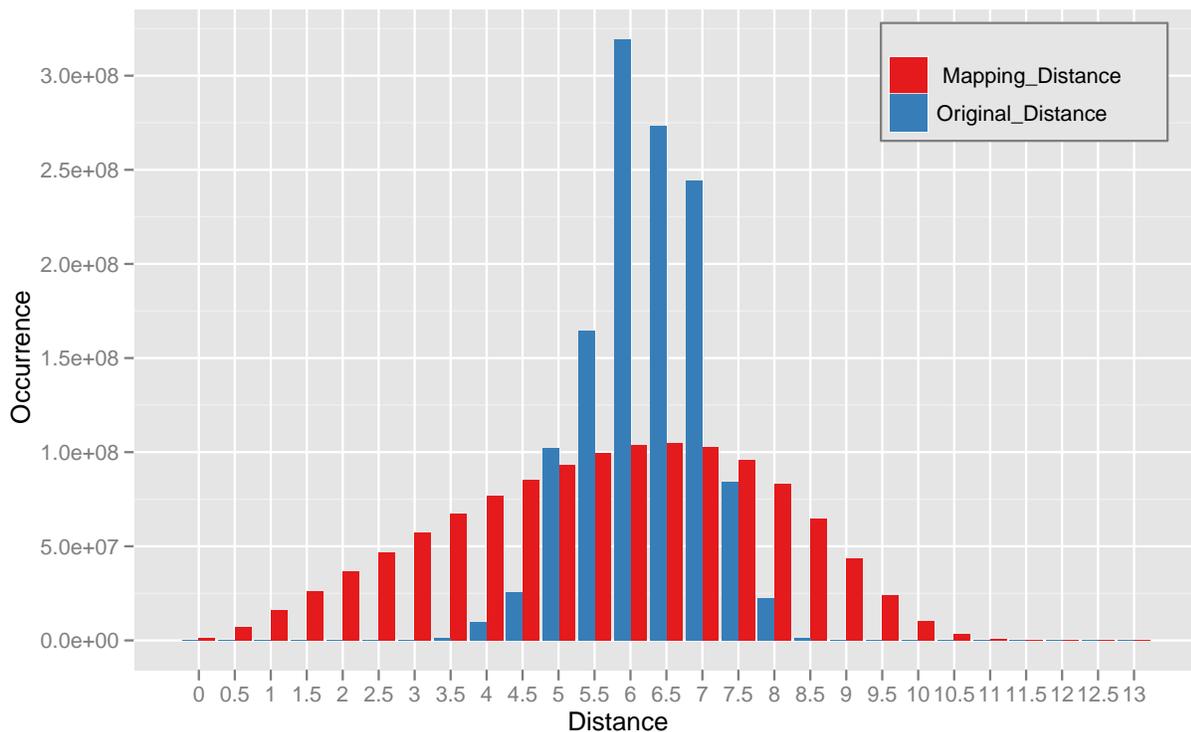
Figure 5: Histogram of the original distance and the pre-mapping distance in the target dimension of 50k sampled data of 100k. The maximum original distance of the 50k sampled data is 10.198 and the maximum mapping distance of the 50k sampled data is 12.960.

the number of the cases of all of the selected nearest neighbors closely mapped is decreased in Table 2. The percentage of the cases of all of the selected neareset neighbors closely mapped is also shown in Table 2. Between the cases of $k = 2$ and $k = 3$, the difference of the all k-NNs closely mapped cases is about 8.2% of a 50k out-of-sample points. For the case of $k = 20$, the occurrence of closely mapped cases is dropped down from 91.8% to 20.6%.

From the above investigation of the mapping distance distribution between selected nearest neighbors, it is found that, even with a small number of nearest neighbors, the neighbors can be mapped relatively far from each other, and the number of those cases is increased as k is increased. The long distance mappings between nearest neighbors could result in generating center-biased mappings by interpolation. We can think of this as a reason for why the 2-NN case shows better results than other cases, which use the larger number of nearest neighbors,

with the Pubchem dataset.

In short, as we explored the optimal number of nearest neighbors with Pubchem data set, $k = 2$ is the optimal case as shown in Fig. 2 with the Pubchem dataset, and the larger nearest neighbor cases show biased mapping results, as shown in Fig. 4. Therefore, we use 2-NN for the forthcoming MI-MDS experimental analyses with Pubchem datasets in this section.

5.2. Comparison between MDS and MI-MDS

5.2.1. Fixed Full Data Case

Fig. 6 shows the comparison of quality between MI-MDS results of 100k data and the MDS (SMACOF) only results with the same 100k Pubchem dataset. The MI-MDS result of 100k dataset is experimented based on different same data sizes, i.e. 12.5k, 25k, and 50k. The y-axis of the plot is the normalized STRESS value which is shown in Eq. (19). The normalized STRESS difference between the MDS only results and interpo-
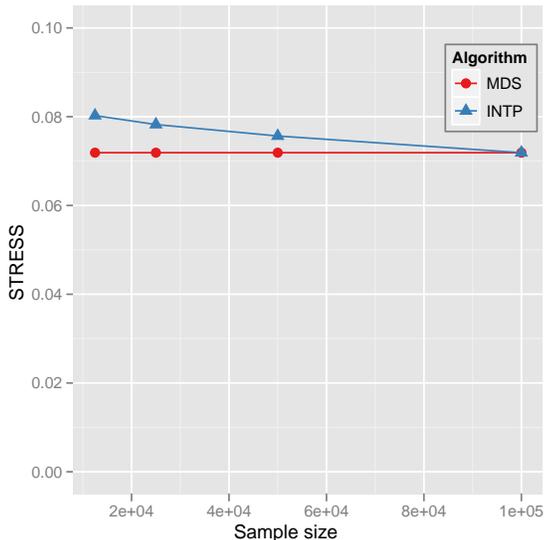
Figure 6: Quality comparison between the interpolated result of 100k with respect to the different sample sizes (INTP) and the 100k MDS result (MDS)
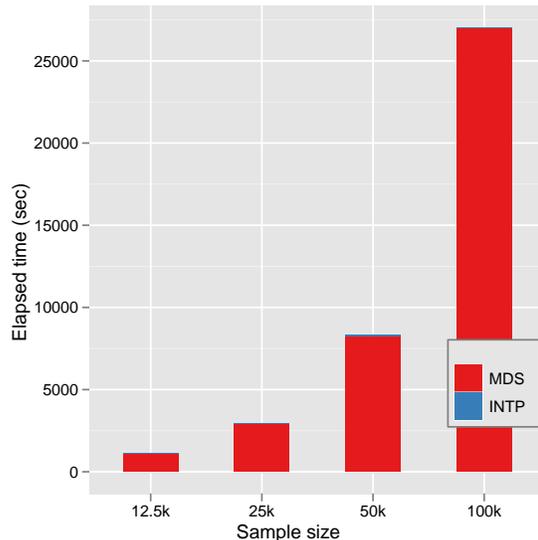

Figure 7: Running time comparison between the Out-of-Sample approach which combines the full MDS running time with sample data and the MI-MDS running time with out-of-sample data when $N = 100k$, with respect to the different sample sizes and the full MDS result of the 100k data.

lated results with 50k is only around 0.0038. Even with a small portion of sample data (12.5k data is only 1/8 of 100k), the proposed MI-MDS algorithm produces good enough mapping in the target dimension using a much smaller amount of time than when we ran MDS with a full 100k of data.

In Fig. 7, we compare the commulated running time of the *out-of-sample* approach, which combines the full MDS running time of sample data and MI-MDS running time of the out-of-sample data with respect to different sample size, to the running time of the full MDS run with the 100k data. As shown in Fig. 7, the overall running time of the out-of-sample approach is much smaller than the full MDS approach. To be more specific, the out-of-sample approach for 100k dataset takes around 25.1, 9.3, and 3.3 times faster than the full MDS approach with respect to different sample sizes, 12.5k, 25k, and 50k, correspondingly.

Fig. 8 shows the MI-MDS interpolation running time only with respect to the sample data using 16 nodes of the Cluster-II in Table 1. The MI-MDS algorithm takes around 8.55, 14.35, and 18.98 seconds with different sample sizes, i.e. 12.5k, 25k, and 50k, to find new mappings of 87500, 75000, and 50000 points based on the pre-mapping results of the corresponding sample data. In Fig. 8, we can find the interesting feature that it takes much less time to find new mappings of 87,500 points (8.55 seconds) than to find new mappings of 50,000 points (18.98 seconds). The reason is the computational time complexity of MI-MDS is $O(Mn)$ where n is the sample size and $M = N - n$. Thus, the running time of MI-MDS is proportional to the number of new mapping points if the sample size (n) is the same, as in the larger data set case shown below in this paper. However, the above case is the opposite case. The full data size (N) is fixed, so that both the sample data size (n) and the out-of-sample data size (M) are variable and correlated. We can illustrate $O(Mn)$ as a simple quadratic equation of variable n as

following: $O(n * (N - n)) = O(N * n - n^2)$, which has maximum when $n = N/2$. The above experiment case $N = 100k$ and $n = 50k$ is the maximum case, so that the case of 50k sample data of MI-MDS took longer than the case of the 12.5k sample data.
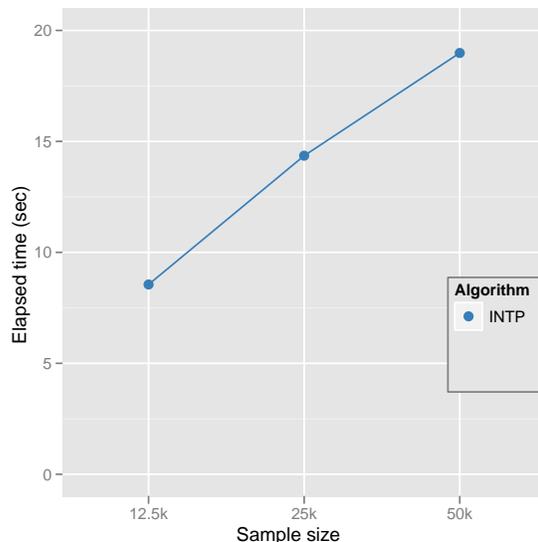

Figure 8: Elapsed time of parallel MI-MDS running time of 100k data with respect to the sample size using 16 nodes of the Cluster-II in Table 1. Note that the computational time complexity of MI-MDS is $O(Mn)$ where n is the sample size and $M = N - n$.

5.2.2. Fixed Sample Data Size

Above we discussed the MI-MDS quality of the fixed total number (100k) with respect to the different sample data sizes,

Table 3: Large-scale MI-MDS running time (seconds) with 100k sample data

1 Million	2 Million	4 Million
731.1567	1449.1683	2895.3414

compared to the MDS running results with total number of data (100k). Now, the opposite direction of the test, which tests the scalability of the proposed interpolation algorithm, is performed as follows: we fix the sample data size to 100k, and the interpolated data size is increased from one million (1M) to two million (2M) and four million (4M). Then, the STRESS value is measured for each running result of total data, i.e. 1M + 100k, 2M + 100k, and 4M + 100k. The measured STRESS value is shown in Fig. 9. There is some quality lost between the full MDS running results with 100k data and the 1M interpolated results based on that 100k mapping; they have about a 0.007 difference in the normalized STRESS criteria. However, there is not much difference between the normalized STRESS values of the 1M, 2M, and 4M interpolated results, although the sample size is quite a small portion of the total data and the out-of-sample data size increases as quadruple the numbers. From the above results, we could consider that the proposed MI-MDS algorithm works well and is scalable if we are given a good enough pre-configured result which represents well the structure of the given data. Note that it is not possible to run the SMACOF algorithm with only 200k data points due to memory bounds, within the systems in Table 1.
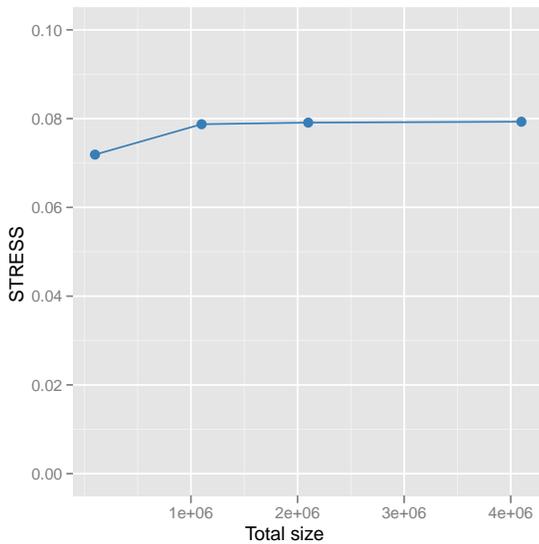

Figure 9: The STRESS value change of the interpolation larger data, such as 1M, 2M, and 4M data points, with 100k sample data. The initial STRESS value of MDS result of 100k data is 0.0719.

We also measure the runtime of the MI-MDS algorithm with a large-scale data set up to 4 million points. Fig. 10 shows the running time of the out-of-sample approach in a commulated bar graph, which represents the full MDS running time of sample data ($M = 100$k) in the red bar and the MI-MDS interpolation time of out-of-sample data ($n = 1$M, 2M, and 4M) in
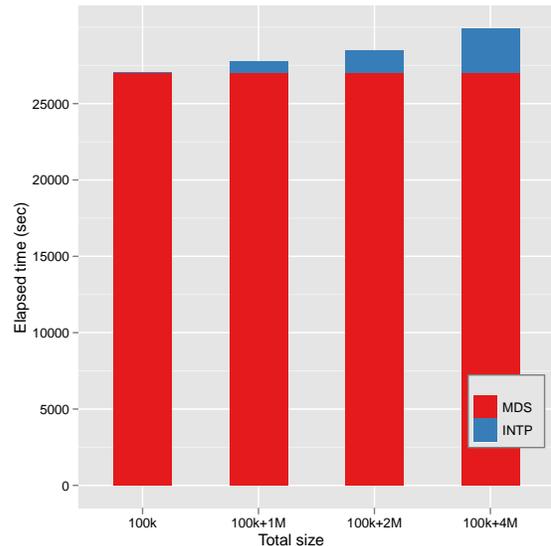

Figure 10: Running time of the Out-of-Sample approach which combines the full MDS running time with sample data ($M = 100k$) and the MI-MDS running time with different out-of-sample data sizes, i.e. 1M, 2M, and 4M.

the blue bar on top of the red bar. As we expected, the running time of MI-MDS is much faster than the full MDS running time in Fig. 10. Although the MI-MDS interpolation running time in Table 3 is much smaller than the full MDS running time (27006 seconds), the MI-MDS deals with a much larger amount of points, i.e. 10, 20, and 40 times larger number of points. Note that we cannot run the parallel SMACOF algorithm [7] with even 200,000 points on our current sytsems in Table 1. Even though we assume that we are able to run the parallel SMACOF algorithm with millions of points on **Cluster-II** in Table 1, the parallel SMACOF will take 100, 400, and 1600 times longer with 1M, 2M, and 4M data than the running time of parallel SMACOF with 100k data, due to the $O(N^2)$ computational complexity. As opposed to the approximated full MDS running time, the proposed MI-MDS interpolation takes much less time to deal with millions of points than parallel SMACOF algorithm. In numeric, MI-MDS interpolation is faster than approximated full parallel MDS running time in 3693.5, 7454.2, and 14923.8 times with 1M, 2M, and 4M data, correspondingly.

If we extract the MI-MDS running time only with respect to the out-of-sample data size from Fig. 10, the running time should be proportional to the number of out-of-sample data since the sample data size is fixed. Table 3 shows the exact running time of the MI-MDS interpolation method with respect to the number of the out-of-sample data size (n), based on the same sample data ($M = 100$k). The running time is almost exactly proportional to the out-of-sample data size (n), as it should be.

5.3. Parallel Performance Analysis of MI-MDS

In the above section, we discussed the quality of the constructed configuration of the MI-MDS approach based on the STRESS value of the interpolated configuration, and the run-

ning time benefits of the proposed MI-MDS interpolation approach. Here, we would like to investigate the MPI communication overhead and parallel performance of the proposed parallel MI-MDS implementation in Section 4.1 in terms of efficiency with respect to the running results within Cluster-I and Cluster-II in Table 1.

First of all, we prefer to investigate the parallel overhead, especially the MPI communication overhead, which could be major parallel overhead for the parallel MI-MDS in Section 4.1. Parallel MI-MDS consists of two different computations, the interpolation part and the STRESS calculation part. The interpolation part is pleasingly parallel and its computational complexity is $O(M)$, where $M = N - n$, if the sample size n is considered as a constant. The interpolation part uses only two MPI primitives, `MPI_GATHER` and `MPI_BROADCAST`, at the end of interpolation to gather all the interpolated mapping results and spread out the combined interpolated mapping results to all the processes for further computation. Thus, the communicated message amount through MPI primitives is $O(M)$, so it is not dependent on the number of processes but the number of whole out-of-sample points.

For the STRESS calculation part, that were applied to the proposed symmetric pairwise computation in Section 4.2, each process uses `MPI_SENDRECV` k times to send an assigned block or rolled block, whose size is M/p, where $k = \lceil (p-1)/2 \rceil$ for communicating required data and `MPI_REDUCE` twice for calculating $\sum_{i<j}(d_{ij}-\delta_{ij})^2$ and $\sum_{i<j}\delta_{ij}^2$. Thus, the MPI communicated data size is $O(M/p \times p) = O(M)$ without regard to the number of processes.

The MPI overhead during the interpolation part and the STRESS calculating part at Cluster-I and Cluster-II in Table 1 are shown in Fig. 11 and Fig. 12, correspondingly. Note that the x-axis of both figures is the sample size (n) but not $M = N - n$. In the figures, the model is generated as $O(M)$ starting with the smallest sample size, here 12.5k. Both Fig. 11 and Fig. 12 show that the actual overhead measurement follows the MPI communication overhead model.

Fig. 13 and Fig. 14 illustrate the efficiency of the interpolation part and the STRESS calculation part of the parallel MI-MDS running results with different sample size - 12.5k, 25k, and 50k - with respect to the number of parallel units using Cluster-I and Cluster-II, correspondingly. Equations for the efficiency is follows:

$$f = \frac{pT(p) - T(1)}{T(1)} \qquad (20)$$

$$\varepsilon = \frac{1}{1 + f} \qquad (21)$$

where p is the number of parallel units, $T(p)$ is the running time with p parallel units, and $T(1)$ is the sequential running time. In practice, Eq. (20) can be replaced with following:

$$f = \frac{\alpha T(p_1) - T(p_2)}{T(p_2)} \qquad (22)$$

where $\alpha = p_1/p_2$ and p_2 is the smallest number of used cores
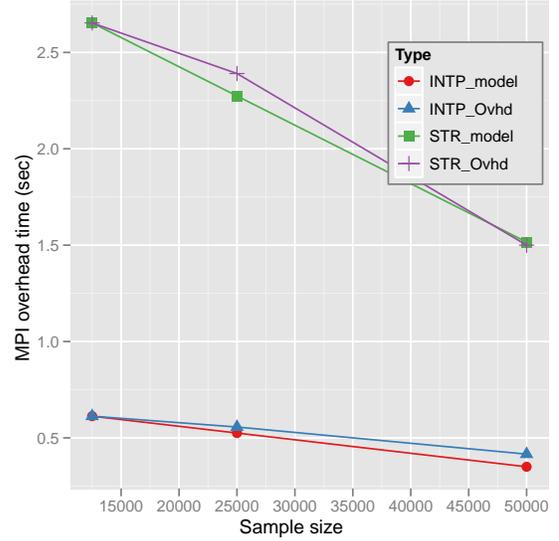

Figure 11: Parallel overhead modeled as due to MPI communication in terms of sample data size (m) using Cluster-I in Table 1 and message passing overhead model.
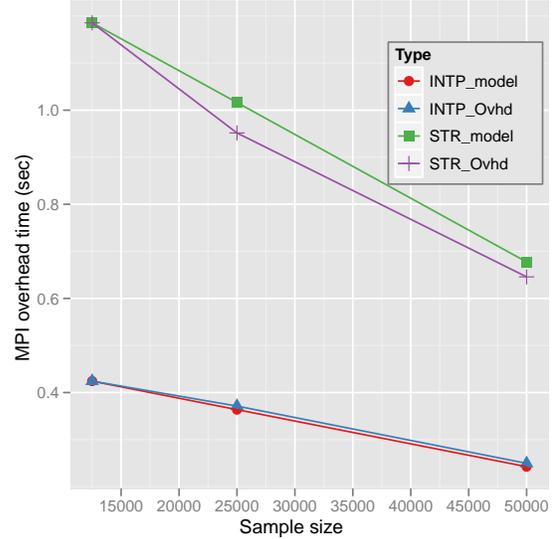

Figure 12: Parallel overhead modeled as due to MPI communication in terms of sample data size (m) using Cluster-II in Table 1 and message passing overhead model.

for the experiment, so $alpha \geq 1$. We use Eq. (22) for the overhead calculation.

In Fig. 13, 16 to 128 cores are used to measure parallel performance with 8 processes, and 32 to 384 cores are used to evaluate the parallel performance of the proposed parallel MI-MDS with 16 processes in Fig. 14. Processes communicate via MPI primitives and each process is also parallelized at the thread level. Both Fig. 13 and Fig. 14 show very good efficiency with an appropriate degree of parallelism. Since both the interpolation part and the STRESS calcualtion part are pleasingly parallel within a process, the major overhead portion is the MPI
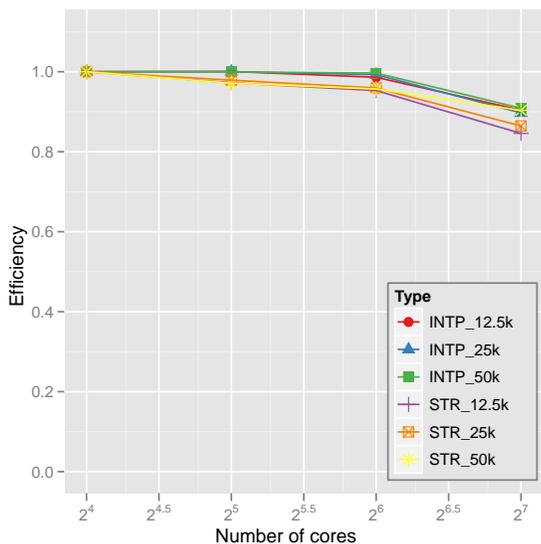
Figure 13: Efficiency of the interpolation part (INTP) and the STRESS evaluation part (STR) runtimes in the parallel MI-MDS application with respect to different sample data sizes using Cluster-I in Table 1. The total data size is 100k.
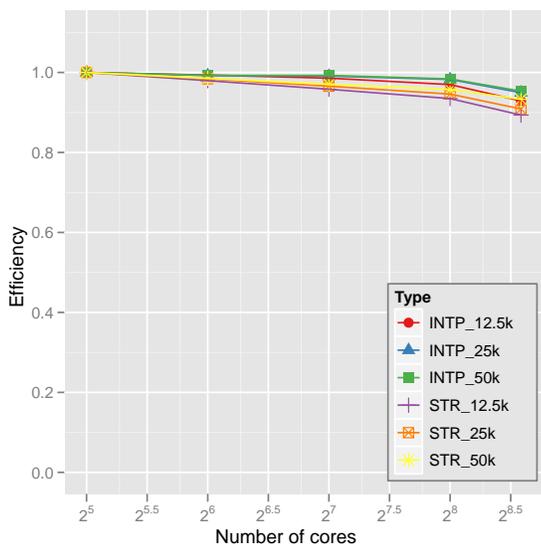

Figure 14: Efficiency of the interpolation part (INTP) and the STRESS evaluation part (STR) runtimes in the parallel MI-MDS application with respect to different sample data sizes using Cluster-II in Table 1. The total data size is 100k.

message communicating overhead unless load balance is not achieved in the thread-level parallelization within each process.

In the previous paragraphs, the MPI communicating overhead is investigated and the MPI communicating overhead shows $O(M)$ relation. Thus, the MPI overhead is constant if we examine it with the same number of processes and the same out-of-sample data sizes. Since the parallel computation time decreases as more cores are used, but the overhead time remains constant, this property lowers the efficiency as the number of cores is increased, as we expected. Note that the number of

processes which lowers the efficiency dramatically is different between the Cluster-I and Cluster-II. The reason is that the MPI overhead time of Cluster-I is bigger than that of Cluster-II due to different network environments, i.e. Giga bit ethernet and 20Gbps Infiniband. The difference is easily found by comparing Fig. 11 and Fig. 12.

5.4. Large-Scale Data Visualization via MI-MDS

Fig. 15 shows the proposed MI-MDS results of a 100k Pub-Chem dataset with respect to the different sample sizes, such as (a) 12.5k and (b) 50k. Sampled data and interpolated points are colored in red and blue, correspondingly. With our parallel interpolation algorithms for MDS, we have also processed a large volume of PubChem data by using our Cluster-II, and the results are shown in Fig. 16. We performed parallel MI-MDS to process datasets of hundreds of thousand and up to 4 million by using the 100k PubChem data set as a training set. In Fig. 16, we show the MI-MDS result of 2 million dataset based on 100k training set, compared to the mapping of 100k training set data. The interpolated points are colored in blue, while the training points are in red. As one can see, our interpolation algorithms have produced a map closed to the training dataset.

6. Conclusion and Future Work

In this paper, we have proposed interpolation algorithms for extending the MDS dimension reduction approaches to very large datasets, i.e up to the millions. The proposed interpolation approach consists of two-phases: (1) the full MDS running with sampled data (n); and (2) the interpolation of out-of-sample data ($N - n$) based on mapped position of sampled data. The proposed interpolation algorithm reduces the computational complexity of the MDS algorithm from $O(N^2)$ to $O(n \times (N - n))$. The iterative majorization method is used as an optimization method for finding mapping positions of the interpolated point. We have also proposed in this paper the usage of parallelized interpolation algorithms for MDS which can utilize multicore/multiprocess technologies. In particular, we utilized a simple but highly efficient mechanism for computing the symmetric all-pairwise distances to provide improved performance.

Before starting a comparative experimental analysis between MI-MDS and the full MDS algorithm, we explored the optimal number of k-NN. 2-NN is the best case for the Pubchem data which we used as a test dataset in this paper. We have shown that our interpolation approach gives results of good quality with high parallel performance. In a quality comparison, the experimental results shows that the interpolation approach output is comparable to the normal MDS output, while it takes much less running time and requires much less memory amounts than the normal MDS methods. The proposed interpolation algorithm is easy to parallelize since each interpolated points is independent to the other out-of-sample points, so many points can be interpolated concurrently without communication. The parallel performance is analyzed in Section 5.3, and it shows very high efficiency as we expected.
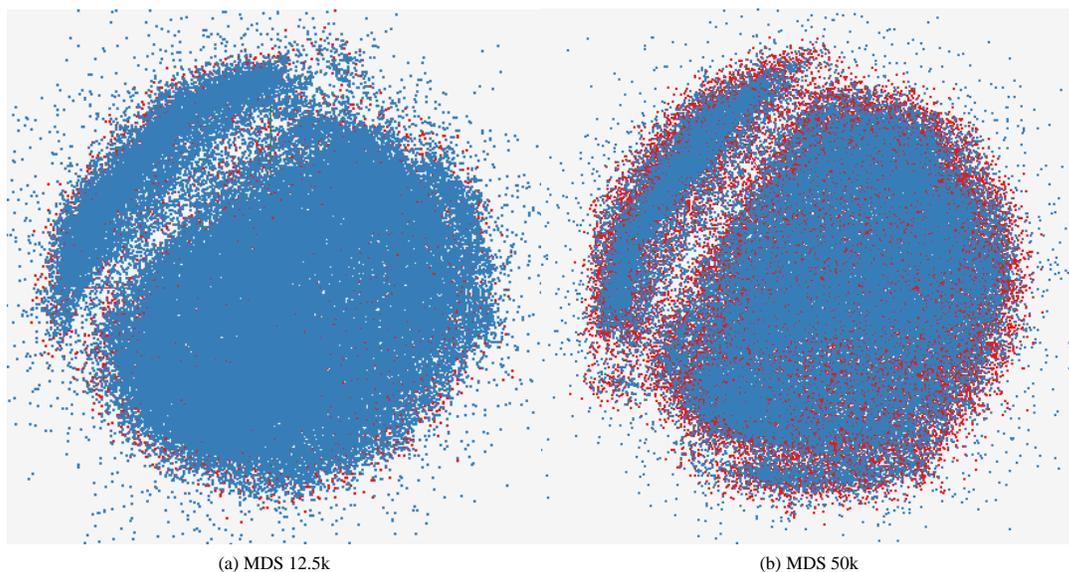
(a) MDS 12.5k   (b) MDS 50k

Figure 15: Interpolated MDS results of total 100k PubChem dataset trained by (a) 12.5k and (b) 50k sampled data. Sampled data are colored in red and interpolated points are in blue.

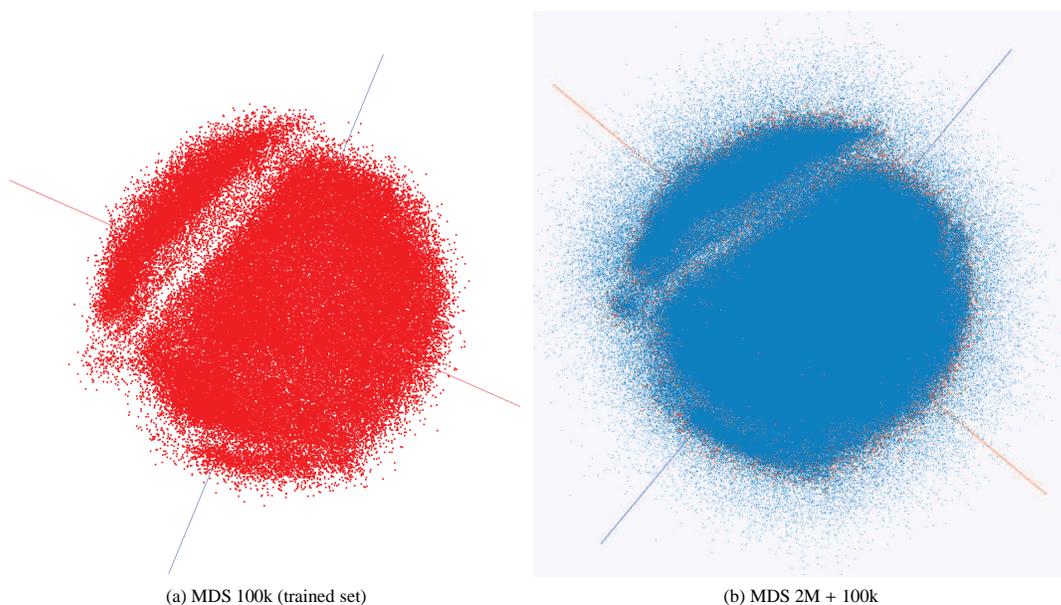(a) MDS 100k (trained set)   (b) MDS 2M + 100k

Figure 16: Interpolated MDS results. Based on 100k samples (a), additional 2M PubChem dataset is interpolated (b). Sampled data are colored in red and interpolated points are in blue.

Consequently, the interpolation approach enables us to configure 4 millions Pubchem data points in this paper with an acceptable normalized STRESS value, compared to the normalized STRESS value of 100k sampled data in less than ONE hour, and the size can be extended further with a moderate running time. If we use parallel MDS only, we cannot even run with only 200,000 points on the Cluster-II system in Table 1 due to the infeasible amount of the memory requirement. Even if it were possible to run parallel MDS with 4 million data points on the Cluster-II system, it would take around 15,000 times longer than the interpolation approach as mentioned in Section 5.2.

Future research could be the application of these ideas to different areas including metagenomics and other DNA sequence visualization. Also, we are working on how to reduce the quality gap between the normal MDS methods and the MDS interpolating method.

References

[1] G. Fox, S. Bae, J. Ekanayake, X. Qiu, H. Yuan, Parallel data mining from multicore to cloudy grids, in: Proceedings of HPC 2008 High Performance Computing and Grids workshop, Cetraro, Italy, 2008.

[2] C. Bishop, M. Svensén, C. Williams, GTM: A principled alternative to the self-organizing map, Advances in neural information processing systems (1997) 354–360.

[3] C. Bishop, M. Svensén, C. Williams, GTM: The generative topographic mapping, Neural computation 10 (1) (1998) 215–234.

[4] T. Kohonen, The self-organizing map, Neurocomputing 21 (1-3) (1998) 1–6.

[5] J. B. Kruskal, M. Wish, Multidimensional Scaling, Sage Publications Inc., Beverly Hills, CA, U.S.A., 1978.

[6] I. Borg, P. J. Groenen, Modern Multidimensional Scaling: Theory and Applications, Springer, New York, NY, U.S.A., 2005.

[7] J. Y. Choi, S.-H. Bae, X. Qiu, G. Fox, High performance dimension reduction and visualization for large high-dimensional data analysis, in: Proceedings of the 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID) 2010, 2010, pp. 331–340.

[8] S.-H. Bae, J. Y. Choi, X. Qiu, G. Fox, Dimension reduction and visualization of large high-dimensional data via interpolation, in: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC) 2010, Chicago, Illinois, 2010, pp. 203–214.

[9] Z. Wang, S. Zheng, Y. Ye, S. Boyd, Further relaxations of the semidefinite programming approach to sensor network localization, SIAM Journal on Optimization 19 (2) (2008) 655–673. doi:http://dx.doi.org/10.1137/060669395.

[10] Y. Bengio, J.-F. Paiement, P. Vincent, O. Delalleau, N. L. Roux, M. Ouimet, Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering, in: Advances in Neural Information Processing Systems, MIT Press, 2004, pp. 177–184.

[11] M. W. Trosset, C. E. Priebe, The out-of-sample problem for classical multidimensional scaling, Computational Statistics and Data Analysis 52 (10) (2008) 4635–4642. doi:http://dx.doi.org/10.1016/j.csda.2008.02.031.

[12] W. S. Torgerson, Multidimensional scaling: I. theory and method, Psychometrika 17 (4) (1952) 401–419.

[13] S. Ingram, T. Munzner, M. Olano, Glimmer: Multilevel mds on the gpu, IEEE Transactions on Visualization and Computer Graphics 15 (2) (2009) 249–261.

[14] J. B. Kruskal, Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis, Psychometrika 29 (1) (1964) 1–27.

[15] Y. Takane, F. W. Young, J. de Leeuw, Nonmetric individual differences multidimensional scaling: an alternating least squares method with optimal scaling features, Psychometrika 42 (1) (1977) 7–67.

[16] J. de Leeuw, Applications of convex analysis to multidimensional scaling, Recent Developments in Statistics (1977) 133–145.

[17] J. de Leeuw, Convergence of the majorization method for multidimensional scaling, Journal of Classification 5 (2) (1988) 163–180.

[18] A. Dempster, N. Laird, D. Rubin, Maximum likelihood from incomplete data via the em algorithm, Journal of the Royal Statistical Society. Series B (1977) 1–38.

[19] T. M. Cover, P. E. Hart, Nearest neighbor pattern classification, IEEE Transaction on Information Theory 13 (1) (1967) 21–27.

[20] J. Qiu, S. Beason, S. Bae, S. Ekanayake, G. Fox, Performance of windows multicore systems on threading and mpi, in: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, IEEE, 2010, pp. 814–819.

[21] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in: International joint Conference on artificial intelligence, Vol. 14, Morgan Kaufmann, 1995, pp. 1137–1145.