Dimension Reduction and Visualization of Large High-Dimensional Data via Interpolation

Seung-Hee Bae
School of Informatics and Computing
Pervasive Technology Institute
Indiana University
Bloomington IN, 47408, USA
sebae@indiana.edu

Jong Youl Choi
School of Informatics and Computing
Pervasive Technology Institute
Indiana University
Bloomington IN, 47408, USA
jychoi@indiana.edu

Judy Qiu
Pervasive Technology Institute
Indiana University
Bloomington IN, 47408, USA
xqiu@indiana.edu

Geoffrey C. Fox
School of Informatics and Computing
Pervasive Technology Institute
Indiana University
Bloomington IN, 47408, USA
gcf@indiana.edu

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 apply high performance dimension reduction algorithms. It facilitates the investigation of unknown structures in a three dimensional visualization. Among the known dimension reduction algorithms, we utilize the multidimensional scaling and generative topographic mapping algorithms to configure the given high-dimensional data into the target dimension. However, both algorithms require large physical memory as well as computational resources. Thus, the authors propose an interpolated approach to utilizing the mapping of only a 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 proposed interpolation algorithms, 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 interpolation method, we construct a configuration of two-million *out-of-sample* data into the target dimension, and the number of *out-of-sample* data can be increased further.

Categories and Subject Descriptors

I.5 [**Pattern Recognition**]: Miscellaneous

General Terms

Algorithms, Performance

Keywords

MDS, GTM, Interpolation

1. INTRODUCTION

Due to the advancements in science and technologies for last several decades, every scientific and technical fields generates a huge amount of data in every minute in the world. We are really in the data deluge era. In reflection of data deluge era, data-intensive scientific computing [12] has been emerging in the scientific computing fields and getting more interested by many people. To analyze those incredible amount of data, many data mining and machine learning algorithms have been developed. Among 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 many dimension reduction algorithms, such as principle component analysis (PCA), generative topographic mapping (GTM) [3,4], self-organizing map (SOM) [16], multidimensional scaling (MDS) [5,18], we discuss about MDS and GTM in this paper since those are popular and theoretically strong. Previously, we parallelize those two algorithms to utilize multicore clusters and to increase the computational capability with minimal overhead for the purpose of investigating large data, such as 100k data [7]. However, parallelization of those algorithms, whose computational complexity and memory requirement is upto $\mathcal{O}(N^2)$ where N is the number of points, is still limited by the memory requirement for huge data, e.g. millions of points, although it utilize 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 interpolation based on pre-configured mappings of the sample data for both MDS and GTM algorithms, so that we can provide configuration of millions points in the target space.

In this paper, first we will briefly discuss about existed methods of *out-of-sample* problem in various dimension reduction algorithms in Section 2. Then, the proposed interpolation methods and how to parallelize them for MDS and GTM algorithms are described in Section 3 and Section 4, correspondingly. The quality comparison between interpolated results and full MDS or GTM running results and parallel performance evaluation of those algorithms are shown in Section 5 followed by our conclusion and future works in Section 6.

2. RELATED WORK

Embedding new points with respect to previously configured points, or known as *out-of-sample* problem, has been actively researched for recent years, aimed at extending the capability of various dimension reduction algorithms, such as LLE, Isomap, multidimensional scaling (MDS), generative topographic mapping (GTM), to name a few. Among many efforts, a recent study by S. Xiang et al. in [24] provides a generalized out-of-sample solutions for non-linear dimension reduction problems by using coodinate propagation. In [6], M. Carreira-Perpiñásn and Z. Lu provides an out-of-sample extension for the algorithms based on the latent variable model, such as generative topographic mapping (GTM), by adapting spectral methods used for Laplacian Eigenmaps.

In 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, semi-definite programming relaxation approaches and its extended approaches has been proposed to solve it [23]. [2] and [21] proposed out-of-sample extension for the classical multidimensional scaling (CMDS) [20], 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. [2] projected the new point $\boldsymbol{x}$ onto the principal components, and [21] extends the CMDS algorithm itself to the out-of-sample problem. Recently, a multilevel force-based MDS algorithm was proposed as well [14].

In contrast to applying out-of-sample problem to CMDS, we extends out-of-sample problem to general MDS results with STRESS criteria in Eq. (1), which finds embeddings of approximating to the distance (or dissimilarity) rather than the inner product as in CMDS, with an EM-like optimization method, called iterative majorizing. The proposed iterative majorizing interpolation approach for the MDS problem will be explained in Section 3.1.

3. MULTIDIMENSIONAL SCALING (MDS)

Multidimensional scaling(MDS) [5, 18] is a general term for the techniques of configuration of the given high dimensional data into target dimensional space based on the pairwise proximity information of the data, while each Euclidean distance between two points becomes as similar to the corresponding pairwise dissimilarity as possible. In other words, MDS is a non-linear optimization problem with respect to mapping in the target dimension and original proximity information.

Formally, the pairwise proximity information is given as an $N \times N$ matrix ($\boldsymbol{\Delta} = [\delta_{ij}]$), where N is the number of points and δ_{ij} is the given dissimilarity value of the original data space between point i and j. (1) Symmetric ($\delta_{ij} = \delta_{ji}$), (2) non-negative ($\delta_{ij} \geq 0$), and (3) zero diagonal ($\delta_{ii} = 0$) are the constraints of the dissimilarity matrix $\boldsymbol{\Delta}$. By MDS algorithm, the generated mapping could be also represented as an $N \times L$ matrix ($\boldsymbol{X}$), where L is the target dimension, and each data point $\boldsymbol{x}_i \in \mathbb{R}^L$ ($i = 1, \ldots, N$) resides in i-th rows of $\boldsymbol{X}$.

The evaluation of the constructed configuration is done with respect to the well-known objective functions of MDS, namely STRESS [17] or SSTRESS [19]. Below equations are the definition of STRESS (1) and SSTRESS (2):

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

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

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

3.1 Majorizing Interpolation MDS

One of the main limitation of most MDS applications is that it requires $\mathcal{O}(N^2)$ memory as well as $\mathcal{O}(N^2)$ computation. Thus, though it is possible to run them with small data size without any trouble, it is impossible to execute it with large number of data due to memory limitation, so it could be considered as memory-bound problem. For instance, Scaling by MAjorizing of COmplicated Function (SMACOF) [9,10], a well-known MDS application via Expectation-Maximization (EM) [11] 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 six $N \times N$ matrices. It is possible to run 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 SMACOF algorithm should have 1.92 TB of memory, which is bigger than total memory of **Cluster-II** in Table 1 (1.536 TB), so it is impossible to run it 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 given data. Our interpolation algorithm is similar to k nearest neighbor (k-NN) classification [8], but we approximate to 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 new mapping position in relation to the k-NN positions, iterative majorization method is used as in SMACOF [9, 10] algorithm, with modified majorization equation, as shown in below. The algorithm proposed in this section is called Majorizing Interpolation MDS (MI-MDS).

The proposed algorithm is implemented as follows. We are given N data in high-dimensional space, say D-dimension, and proximity information ($\boldsymbol{\Delta} = [\delta_{ij}]$) of those data as in Section 3. Among N data, the configuration of the n sample points in L-dimensional space, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^L$, called X, are already constructed by an MDS algorithm, here we use SMACOF algorithm. Then, we select k nearest neighbors, $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_k \in \boldsymbol{P}$, of the given new point among n pre-mapped points with respect to corresponding δ_{ix}, where $\boldsymbol{x}$ represents the new point. Finally, the new mapping of the given new point $\boldsymbol{x} \in \mathbb{R}^L$ is calculated based on the pre-mapped position of selected k-NN and corresponding proximity information δ_{ix}. The finding new mapping position is considered as a minimization problem of STRESS (1) as similar as normal MDS problem with m points, where $m = k + 1$. However, only one point $\boldsymbol{x}$ is movable among m points, so we can summarize STRESS (1) as belows, and we set $w_{ij} = 1$, for $\forall i, j$ in order to simplify.

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

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

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

$$\sum_{i=1}^{k} d_{ix}^2 = \|\boldsymbol{x} - \boldsymbol{p}_1\|^2 + \cdots + \|\boldsymbol{x} - \boldsymbol{p}_k\|^2 \tag{5}$$

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

where $\boldsymbol{q}^t = (\sum_{i=1}^{k} p_{i1}, \ldots, \sum_{i=1}^{k} p_{iL})$ and p_{ij} represents j-th element of $\boldsymbol{p}_i$. 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 [5] for details of how to apply *Cauchy-Schwarz* inequality to $-d_{ix}$. Since $d_{ix} = \|\boldsymbol{p}_i - \boldsymbol{x}\|$, $-d_{ix}$ could have following inequality based on *Cauchy-Schwarz* inequality:

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

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

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

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

$$= -\boldsymbol{x}^t \sum_{i=1}^{k} \frac{\delta_{ix}}{d_{iz}} (\boldsymbol{z} - \boldsymbol{p}_i) + \mathcal{C}_\rho \tag{10}$$

where $\mathcal{C}_\rho$ is a constant. If Eq. (6) and Eq. (10) are applied to Eq. (4), then it could be like following:

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

$$\leq \mathcal{C} + k\|\boldsymbol{x}\|^2 - 2\boldsymbol{x}^t\boldsymbol{q} + \sum_{i=1}^{k} \|\boldsymbol{p}_i\|^2$$

$$- 2\boldsymbol{x}^t \sum_{i=1}^{k} \frac{\delta_{ix}}{d_{iz}} (\boldsymbol{z} - \boldsymbol{p}_i) + \mathcal{C}_\rho \tag{12}$$

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

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

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

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

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

The advantage of the iterative majorization algorithm is that it produces a series of mapping with non-increasing STRESS value as proceeds, which results in local minima. It is good enough to find local minima, since the proposed MI 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 $\boldsymbol{z}$ with $\boldsymbol{x}^{[t-1]}$ in Eq. (15), then we generate an iterative majorizing equation like following:

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

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

where $d_{iz} = \|\boldsymbol{p}_i - \boldsymbol{x}^{[t-1]}\|$ and $\overline{\boldsymbol{p}}$ 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 same as that of SMACOF algorithm, which is

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

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

Process of the out-of-sample MDS could be summarized as 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.

The summary of proposed MI algorithm for interpolation of a new data, say $\boldsymbol{x}$, in relation to pre-mapping result of the

Algorithm 1 Majorizing Interpolation (MI) algorithm

```
Find $k$-NN: find $k$ nearest neighbors of $\boldsymbol{x}$, $\boldsymbol{p}_i \in \boldsymbol{P}$ $i = 1, \ldots, k$ of the given new data based on original dissimilarity $\delta_{ix}$.
Gather mapping results in target dimension of the $k$-NN.
Calculate $\overline{\boldsymbol{p}}$, the average of pre-mapped results of $\boldsymbol{p}_i \in \boldsymbol{P}$.
Generate initial mapping of $\boldsymbol{x}$, called $\boldsymbol{x}^{[0]}$, either $\overline{\boldsymbol{p}}$ or a random point.
Compute $\sigma(\boldsymbol{S}^{[0]})$, where $\boldsymbol{S}^{[0]} = \boldsymbol{P} \cup \{\boldsymbol{x}^{[0]}\}$.
while $t = 0$ or $(\Delta\sigma(\boldsymbol{S}^{[t]}) > \varepsilon$ and $t \le$ MAX_ITER$)$ do
    increase $t$ by one.
    Compute $\boldsymbol{x}^{[t]}$ by Eq. (17).
    Compute $\sigma(\boldsymbol{S}^{[t]})$.
end while
return $\boldsymbol{x}^{[t]}$;
```

sample data is described in Alg. 1. Note that the algorithm uses $\overline{\boldsymbol{p}}$ as an initial mapping of the new point $\boldsymbol{x}^{[0]}$ unless initialization with $\overline{\boldsymbol{p}}$ makes $d_{ix} = 0$, since the mapping is based on the k-NN. If $\overline{\boldsymbol{p}}$ makes $d_{ix} = 0$ $(i = 1, \ldots, k)$, then we use a random generated point as an initial position of $\boldsymbol{x}^{[0]}$.

3.2 Parallel MI-MDS Algorithm

Suppose that, among N points, 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. $\mathcal{O}(Mn+n^2)$ vs. $\mathcal{O}(N^2)$, implementing parallel MI algorithm is essential, since M can be still huge, like millions. In addition, most of clusters are now in forms of multicore-clusters after multicore-chip invented, so we are using hybrid-model parallelism, which combine processes and threads together.

In contrast to the original MDS algorithm that 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 of among interpolated points makes the MI-MDS algorithm to be pleasingly-parallel. In other words, there must be minimum communication overhead and 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 each other.

3.3 Parallel Pairwise Computation with Subset of Data

Although interpolation approach itself is in $\mathcal{O}(Mn)$, if we want to evaluate the quality of the interpolated results by STRESS criteria in Eq. (1) of overall N points, it requires $\mathcal{O}(N^2)$ 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 well as the information of all pre-mapped n points. It is natural way of using distributed-memory system, such as cluster systems, to access only 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 necessary data to compute STRESS.

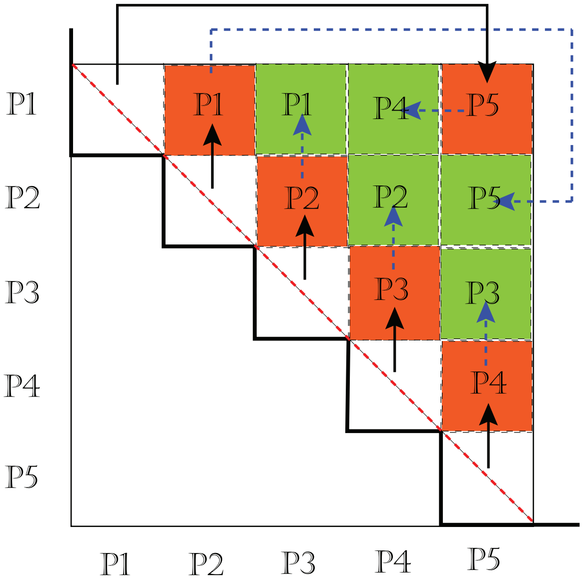


Figure 1: Message passing pattern and parallel symmetric pairwise computation for calculating STRESS value of whole mapping results.

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

Algorithm 2 Parallel Pairwise Computation

```
input: $\boldsymbol{Y}$ = a subset of data;
input: $p$ = the number of process;
$rank \Leftarrow$ the rank of process;
$sendTo \Leftarrow (rank - 1)$ mod $p$
$recvFrom \Leftarrow (rank + 1)$ mod $p$
$k \Leftarrow 0$;
Compute upper triangle in the diagonal blocks in Fig. 1;
$MAX\_ITER \Leftarrow \lceil (p-1)/2 \rceil$
while $k < MAX\_ITER$ do
    $k \Leftarrow k + 1$;
    if $k = 1$ then
        MPI_SENDRECV($\boldsymbol{Y}, sendTo, \boldsymbol{Y}_r, recvFrom$);
    else
        $\boldsymbol{Y}_s \Leftarrow \boldsymbol{Y}_r$;
        MPI_SENDRECV($\boldsymbol{Y}_s, sendTo, \boldsymbol{Y}_r, recvFrom$);
    end if
    Do Computation();
end while
```

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 received data block instead of the originally assigned data block. The green blocks and dotted blue arrows show the 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 odd number case and even number case is that two processes are assigned to one block at the last iteration of even number case, but not in odd number case. Though two processes are assigned to single block, it is easy to achieve load balance by dividing two section of the block and assign them to each process. Therefore, both odd number processes and even number processes cases are parallelized well using the above rolling-computing scheme, with minimal message passing overhead. The summary of the above parallel pairwise computation is shown in Alg. 2.

4. GENERATIVE TOPOGRAPHIC MAPPING

The Generative Topographic Mapping (GTM) algorithm has been developed to find an optimal representation of high-dimensional data in the low-dimensional space, or also known as latent space. Unlike the well-known PCA-based dimension reduction which finds linear embeddings in the target space, the GTM algorithm seeks a non-linear mappings in order to provide more improved separations than PCA [4]. Also, in contrast to Self-Organized Map (SOM) which finds lower dimensional representations in a heuristic approach with no explicit density model for data, the GTM algorithm finds a specific probability density based on Gaussian noise model. For this reason, GTM is often called as a principled alternative to SOM [3].

In GTM algorithm, one seeks a non-linear mapping of user-defined K points $\{\boldsymbol{z}_i\}_{i=1}^{K}$ in the latent space to the original data space for N data points in a way K data points can optimally represent N data points $\{\boldsymbol{x}_j\}_{j=1}^{N}$ in the high-dimensional space. More specifically, the GTM algorithm finds a non-linear mapping $f(\boldsymbol{z}_i; \boldsymbol{W})$ with a weight parameter set $\boldsymbol{W}$ and a coefficient β which maximize the following log-likelihood:

$$\mathcal{L}(\boldsymbol{W}, \beta) = \sum_{j=1}^{N} \ln \left\{ \frac{1}{K} \sum_{i=1}^{K} \mathcal{N}(\boldsymbol{x}_j | f(\boldsymbol{z}_i; \boldsymbol{W}), \beta) \right\}, \qquad (19)$$

where $\mathcal{N}(\boldsymbol{a}|\boldsymbol{b}, \beta)$ represents Gaussian probability for $\boldsymbol{a}$ centered on $\boldsymbol{b}$ with variance β^{-1} (known as precision).

This problem is a variant of well-known K-clustering problem which is NP-hard [1]. To solve the probem, GTM algorithm uses Expectation-Maximized (EM) method to find a local optimal solution. Since the details of GTM algorithm is out of this paper's scope, we recommend readers to refer to the original GTM papers [3, 4] for more details.

Once found an optimal parameter set in the GTM algorithm, we can draw a GTM map (also known as posterior mean projection plot) for N data points in the latent space by using the following equation:

$$\langle \boldsymbol{x}_j \rangle \quad = \quad \sum_{i=1}^{K} r_{ij} \boldsymbol{z}_i \qquad (20)$$

where r_{ij} is the posterior probabilities, called *responsibilities*, defined by

$$r_{ij} \quad = \quad \frac{\mathcal{N}(\boldsymbol{x}_j | \boldsymbol{y}_i, \beta)}{\sum_{i'=1}^{K} \mathcal{N}(\boldsymbol{x}_j | \boldsymbol{y}_{i'}, \beta)} \qquad (21)$$

for $\boldsymbol{y}_i = f(\boldsymbol{z}_i; \boldsymbol{W})$

4.1 GTM Interpolation

The core of GTM algorithm is to find the best K representations for N data points, which makes the problem complexity is $\mathcal{O}(KN)$. Since in general $K \ll N$, the problem is sub $\mathcal{O}(N^2)$ which is the case of the MDS problem. However, for large N the computations in GTM algorithm is still challenging. For example, to draw a GTM map for 0.1 million 166-dimensional data points in a 20x20x20 latent grid space, in our rough estimation it took about 30 hours by using 64 cores.

To reduce such computational burden, we can use interpolation approach in GTM algorithm, in which one can find the best K representations from a portion of N data points, known as *samples*, instead of processing full N data points and continue to process remaining or *out-of-sample* data by using the information learned from previous samples. Typically, since the latter (interpolation process) doesn't involve computationally expensive learning processes, one can reduce overall execution time compared to the time spent for the full data processing approach.

Although many researches have been performed to solve various non-linear manifold embeddings, including GTM algorithm, with this out-of-sample approach, we have chosen a simple approach since it works well for our dataset used in this paper. For more sophisticated and complexed data, one can see [6, 15, 24].

With a simple interpolation approach, we can perform the GTM process as follows:

1. **Sampling** – randomly select a sample set $\boldsymbol{S} = \{\boldsymbol{s}_k\}_{k=1}^{N'}$ of size N' from the full dataset $\{\boldsymbol{x}_j\}_{j=1}^{N}$ of size N, where $\boldsymbol{s}_k, \boldsymbol{x}_j \in \mathbb{R}^D$.

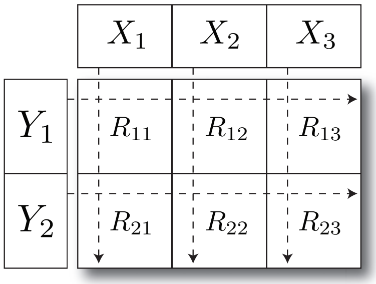


Figure 2: An example of decomposition in GTM interpolation for computing responsibility matrix R in parallel by using a 2x3 virtual processor grid.

2. **GTM process** – perform GTM algorithm and find an optimal K cluster center $\{\boldsymbol{y}_i\}_{i=1}^{K}$ and a coefficient β for the sample set $\boldsymbol{S}$. Let $\boldsymbol{Y} \in \mathbb{R}^{K \times D}$ denote the matrix representation of K centers, where i-th row contains $\boldsymbol{y}_i \in \mathbb{R}^D$.

3. **Compute responsibility** – for the remaining dataset $\{\boldsymbol{t}_n\}_{n=1}^{M}$ of size $M = N - N'$, denoted by $\boldsymbol{T}$, compute a $K \times M$ pairwise distance matrix $\boldsymbol{D}$ where (i, j)-th element d_{ij} is a Gaussian probability between $\boldsymbol{t}_j$ and $\boldsymbol{y}_i$ with variance β^{-1}, which was learned from the samples, such as $d_{ij} = \mathcal{N}(\boldsymbol{t}_j | \boldsymbol{y}_i, \beta)$. Compute a responsibility matrix R by , as follows:

$$\boldsymbol{R} = \boldsymbol{D} \oslash (\boldsymbol{e}\boldsymbol{e}^t \boldsymbol{D}) \tag{22}$$

where a vector $\boldsymbol{e} = (1, ..., 1)^t \in \mathbb{R}^K$ and $\oslash$ represents element-wise division.

4. **Interpolated map** – by using Eq. (20), an interpolated GTM map $\bar{\boldsymbol{Z}}$, known as posterior mean plot, can be computed by the following equation:

$$\bar{\boldsymbol{Z}} = \boldsymbol{R}^t \boldsymbol{Z} \tag{23}$$

where Z denotes the matrix represents of the latent points $\{\boldsymbol{z}_k\}_{k=1}^{K}$.

In the GTM interpolation, computing responsibility matrix R (step 3) is the most time and memory consuming step. This step can be parallelized by decomposing the problem into P-by-Q sub blocks such that total PQ processes can concurrently process each block which approximately holds $1/PQ$ elements of R. This is the same method used in parallel GTM implementation discussed in [7]. More detailed algorithm and analysis will be discussed in Section 4.2

4.2 Parallel GTM Interpolation

The core of parallel GTM interpolation is how to parallelize the computations for the responsibility matrix R as in (22) since its computation is the most time and memory consuming task. This can be done by the same approach for the general parallel matrix multiplication methods known as Fox algorithm [13] and SUMMA [22] but with extra tasks.

Assuming that we have $P \times Q$ virtual compute grids with total $p = PQ$ processes, we can decompose the responsibility matrix R into $P \times Q$ sub blocks so that (u, v)-th sub block of R, denoted by $\boldsymbol{R}_{uv}$ for $u = 1, ..., P$ and $v = 1, ..., Q$, can be computed by one process. For this computation, we also need to decompose the matrix $\boldsymbol{Y}$ for K cluster centers into P sub blocks such that each block contains approximately K/P cluster centers and divide the M out-of-sample data $\boldsymbol{T}$ into Q sub blocks that one block contains approximately M/Q out-of-sample data. Then, we can compute (i, n)-th sub block of $\boldsymbol{R}$ as follows:

$$\boldsymbol{R}_{uv} = \text{resp}(\boldsymbol{Y}_u, \boldsymbol{T}_v) \tag{24}$$

where resp$(\cdot, \cdot)$ is a function for computing responsibility and $\boldsymbol{Y}_u$ and $\boldsymbol{T}_v$ are inputs, denoted by u-th and v-th sub block of $\boldsymbol{Y}$ and $\boldsymbol{T}$ respectively.

A sketch of parallel GTM interpolation is as follows:

1. Broadcast sub-block $\{\boldsymbol{Y}_u\}_{u=1}^{P}$ and $\{\boldsymbol{Z}_u\}_{u=1}^{P}$ to P rank-one node in rows and $\{\boldsymbol{T}_v\}_{v=1}^{Q}$ to each rank-one node in columns of the $P \times Q$ grid respectively.

2. The rank-one node broadcast sub-block $\{\boldsymbol{Y}_u\}_{u=1}^{P}$ and $\{\boldsymbol{Z}_u\}_{u=1}^{P}$ to row members and $\{\boldsymbol{T}_v\}_{v=1}^{Q}$ to its column members respectively.

3. (u, v)-th node computes sub distance matrix $\boldsymbol{D}_{uv}$ by using $\boldsymbol{Y}_u$ and $\boldsymbol{T}_v$.

4. Collect P vectors $\{\boldsymbol{d}_u\}_{u=1}^{P}$ contains column sum of $\boldsymbol{D}_{uv}$ from column members and compute $\boldsymbol{R}_{uv}$ by

$$\boldsymbol{R}_{uv} \quad = \quad \boldsymbol{D}_{uv} \oslash e \sum_{u}^{P} (\boldsymbol{d}_u)^t \tag{25}$$

where a vector $\boldsymbol{e} = (1, ..., 1)^t \in R^{K/P}$ and $\oslash$ represents element-wise division.

5. (u, v)-th node computes the following sub block for the posterior mean $\bar{\boldsymbol{Z}}_{uv}$

$$\bar{\boldsymbol{Z}}_{uv} = (\boldsymbol{R}_{uv})^t \boldsymbol{Z}_{uv} \tag{26}$$

and send to the rank-one node in columns. Then, v-th sub block of GTM map $\tilde{\boldsymbol{Z}}$ for N/Q data points is

$$\tilde{\boldsymbol{Z}}_v = \sum_{u}^{P} \bar{\boldsymbol{Z}}_{uv} \tag{27}$$

In our parallel GTM interpolation algorithm, main computation time is spent in computing $K/P \times N/Q$ distance matrix $\boldsymbol{D}_{uv}$ which requires approximately $\mathcal{O}(KN/PQ)\tau_C$ computation time, where τ_C represents a unit time for basic operations such as multiplication and summation. Regarding the time spent for communication, our algorithm consumes time due to the network bandwidth, denote τ_B, which increases as the size of data to send and receive. Assuming a minimum spanning tree broadcasting and collecting operations, our algorithm will spend $\mathcal{O}(\log P(KD/Q + N/Q + KN/PQ) + \log P(KD/P + KL/P))\tau_B$. Also, as we increase P or Q, an overhead can occur due to the network latency, denote τ_L, which mainly occurs due to the number of communications. In our parallel GTM interpolation, we have $\mathcal{O}(2P + Q)$ communications. Since generally $L \ll D$ and those are constant, we can formulate the total computing time $T(K, N, P, Q)$ in our algorithm, with respect to K latent points, N data points, and $P \times Q$ processes, as follows:

$$\begin{aligned} &\mathcal{O}\left(\frac{KN}{PQ}\right)\tau_C + \mathcal{O}(2P + Q)\tau_L \\ &\qquad + \mathcal{O}\left(\log P\left(\frac{N}{Q} + \frac{KN}{PQ}\right) + \log Q\left(\frac{KD}{P}\right)\right)\tau_B \end{aligned} \tag{28}$$

where τ_C, τ_B, and τ_L represents unit time for computation, network bandwidth, and network latency respectively.

Since the running time with no parallelization $T_1 = \mathcal{O}(KN)$, we can expect the speed up as $S(K,N,P,Q) = T_1/T(K,N,P,Q)$ and its corresponding efficiency $E(K,N,P,Q) = S(K,N,P,Q)/PQ$ as follows.

$$\left\{1 + \mathcal{O}\left(\frac{\log P}{KPQ^2} + \frac{\log P}{P^2Q^2} + \frac{\log Q}{NP^2Q}\right)\tau_B + \mathcal{O}\left(\frac{2P+Q}{PQ}\right)\tau_L\right\}^{-1} \quad (29)$$

The above equation implies the following: in the case where P and Q are large enough but network latency due to τ_L is negligibly small, we can achieve an ideal efficiency $E(K,N,P,Q) = 1$. However, as we increase the number of cores, the efficiency of our algorithm will be hurt by the increase number of communications. Our experiment results support this expectation.

5. ANALYSIS OF EXPERIMENTAL RESULTS

To measure the quality and parallel performance of our MDS and GTM with interpolation approach discussed in this paper, we have used 166-dimensional chemical dataset obtained from PubChem project database[1], which is a 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 randomly selected up to 2 million chemical subsets for our testing. The computing clusters we have used in our experiments are summarized in Table 1.

In the following, we will mainly show i) the quality of our interpolation approaches in performing MDS and GTM algorithms, with respect to various sample sizes – 12.5k, 25k, and 50k randomly selected from 100k dataset as a basis, and ii) performance measurement of our parallelized interpolation algorithms on our clustering systems as listed in Table 1, and finally, iii) our results on processing up to 2 million MDS and GTM maps based on the trained result from 100K dataset.

5.1 Mapping Quality Comparison

5.1.1 MDS vs. MI-MDS

Generally, the quality of k-NN (k-nearest neighbor) classification (or regression) is related to the number of neighbors. For instance, if we choose larger number for the k, then the algorithm shows higher bias but lower variance. On the other hands, the k-NN algorithm shows lower bias but higher variance based on smaller number of neighbors. The purpose of the MI algorithm is to find appropriate embeddings for the new points based on the given mappings of the sample data, so it is better to be sensitive to the mappings of the k-NN of the new point than to be stable with respect to the mappings of whole sample points. Thus, in this paper, the authors use 2-NN for the MI algorithm.

Fig. 3 shows the comparison of quality between interpolated results of 100K data with different sample data size by using 2-NN and MDS (SMACOF) only result with 100k pubchem data. The y-axis of the plot is STRESS (1) normalized with $\sum_{i<j}\delta_{ij}^2$, and the difference between MDS only results and interpolated with 50k is only around 0.004. Even with small portion of sample data (12.5k data is only 1/8 of 100k), the proposed MI algorithm produces good enough mapping in target dimension using very smaller amount of time than when we run MDS with full 100k data. Fig. 4 shows the MI-MDS running time with respect to the sample data using 16 nodes of the Cluster-II in Table 1. Note that the full MDS running time with 100k using 16 nodes of the Cluster-II in Table 1 is around 27006 sec.

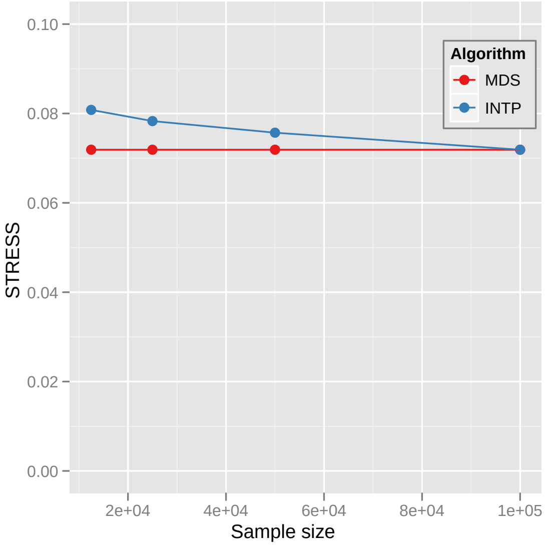


Figure 3: Quality comparison between Interpolated result of 100k with respect to the different sample size (INTP) and 100k MDS result (MDS)

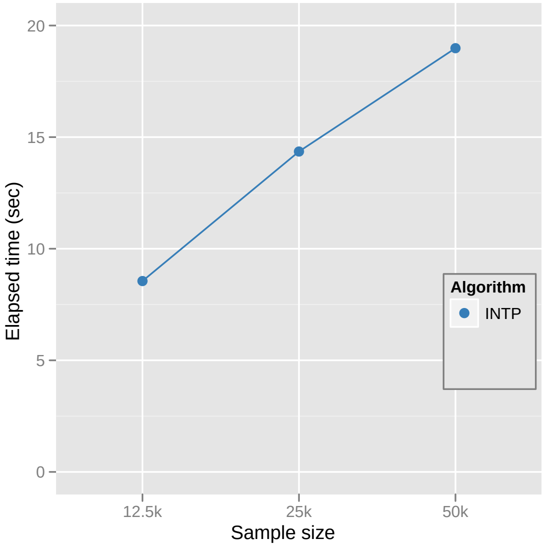


Figure 4: Elapsed time of parallel MI-MDS running time of 100k data with respect to the sample size using 16 nodes of the Cluster-II in Table 1. Note that the computational time complexity of MI-MDS is $\mathcal{O}(Mn)$ where n is the sample size and $M = N - n$.

[1] PubChem,http://pubchem.ncbi.nlm.nih.gov/

Table 1: Compute cluster systems used for the performance analysis

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

Above we discussed about the MI-MDS quality of the fixed total number (100k) and with respect to the different sample data size, compared to MDS running result with total number of data (100k). Now, the opposite direction of test, which tests scalability of the proposed interpolation algorithm, is performed as following: we fix the sample data size to 100k, and the interpolated data size is increased from one millions (1M) to two millions (2M). Then, the STRESS value is measured for each running result of total data, i.e. 1M + 100k and 2M + 100k. The measured STRESS value is shown in Fig. 5. There are some quality lost between the full MDS running result with 100k data and the 1M interpolated results based on that 100k mapping, which is about 0.007 difference in normalized STRESS criteria. However, there is no much difference between the 1M interpolated result and 2M interpolated result, although the sample size is quite small portion of total data and the out-of-sample data size increases as twice. From the above result, we could consider that the proposed MI-MDS algorithm works well and 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 SMACOF algorithm with only 200k data points due to memory bound, within the systems in Table 1.

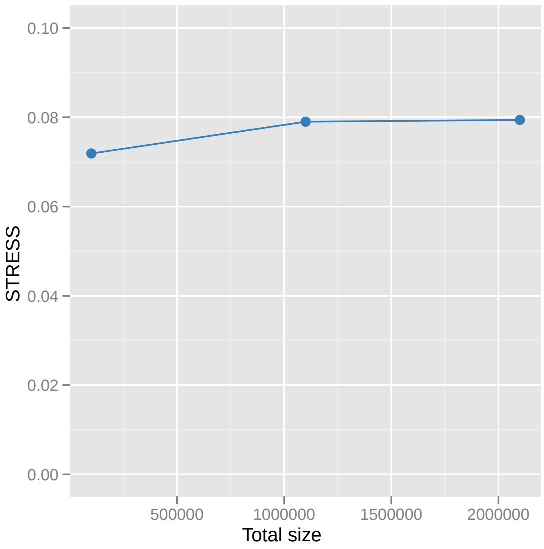


Figure 5: STRESS value change of Interpolation larger data, such as 1M and 2M data points, with 100k sample data. The initial STRESS value of MDS result of 100k data is 0.0719.

5.1.2 GTM vs. Interpolated-GTM

To measure the quality of GTM interpolation algorithm, we have compared quality of GTM maps, generated by using the full GTM processing with no interpolation (hereafter GTM for short) versus maps by using the interpolation approach (hereafter INTP for short) with various sample sizes (12.5k, 25k, and 50k) in terms of maximum log-likelihood (a large maximum log-likelihood implies better quality of map), as defined in (19). Our test result is shown in Fig. 6 where negative log-likelihood values are plotted for each test case and thus points in lower area represent the better quality. Our results show that the interpolation approach has produced almost the same quality of maps with the map generated by using the full dataset. Also the result shows that the interpolation with small sample size (12.5k) is the worst performance case, while others' results are quite close to the the full data processing case. This is the common case in which small samples can lead skewed interpretation of the full data.

Next, we compare GTM versus INTP (GTM with interpolation) in terms of processing time. For this purpose, we processed up to 100k dataset with various sample sizes starting from 12.5k up to 50k and compare the processing time with the case of no use of interpolation (full 100k dataset). Since the original GTM algorithm uses EM method in which a local solution is found in an iterative fashion and the number of iterations is unpredictable, we have measured average number of iterations for processing for the 100k dataset and use this average to project GTM running time for processing 12.5k, 25k, and 50k dataset. By adding interpolation running time as we measured in the previous experiment, we have projected the total running time for using GTM interpolation with various sampling sizes as shown in Fig. 7. As we expected, since the interpolation process has no computing-intensive training processes, the processing time is very short, compared with the approach of using full dataset.

By combining the results shown in Fig. 6 and Fig. 7, we show that GTM interpolation can be used to produce the same quality of GTM maps with much less computing time.

5.2 Parallel Performance

5.2.1 Parallel MI-MDS

In the above section, we discussed about the quality of constructed configuration of MI-MDS based on the STRESS value of the interpolated configuration. Here, we would like to investigate the MPI communication overhead and parallel performance of the proposed parallel MI-MDS implementation in terms of efficiency with respect to the running results

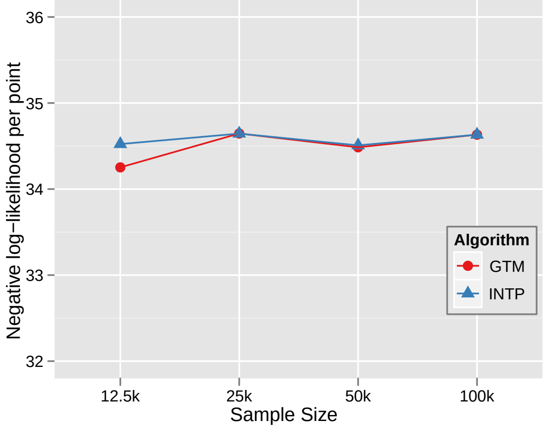


Figure 6: Comparison of maximum log-likelihood from GTM with full-dataset processing without interpolation (GTM) and the GTM with interpolation approach (INTP) with various sampling sizes – 12.5k, 25k, and 50k – for 100k PubChem dataset. As the sample size increases, INTP finds very close results with GTM.

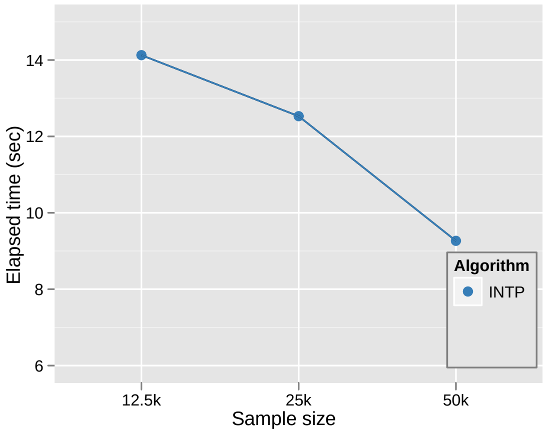


Figure 7: Processing time for GTM interpolation with various sample sizes (12.5k, 25k, and 50k) for 100k PubChem dataset. Elapsed time for 12.5k samples (meaning that we have process 87.5k dataset for interpolation based on 12.5k random samples from the full 100k dataset) is the largest, while 50k samples takes least time. The running time is very small compared with the time for full 100k GTM processing, which generally takes a few hours.

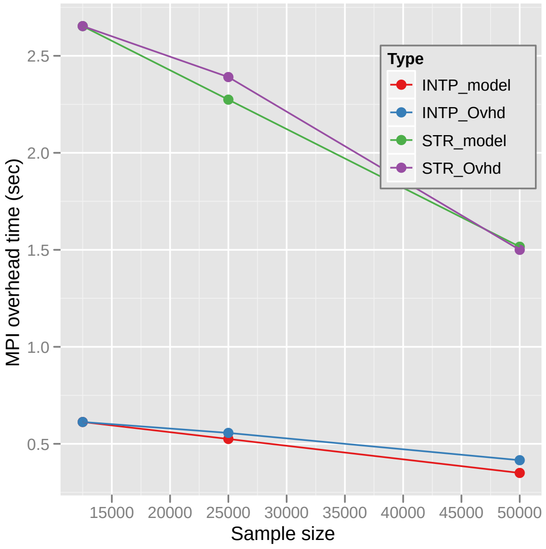


Figure 8: Parallel overhead modeled as due to MPI communication in terms of sample data size (m) using Cluster-I in Table 1 and message passing overhead model.

within Cluster-I and Cluster-II in Table 1.

First of all, we prefer to investigate the parallel overhead, specially MPI communication overhead which could be major parallel overhead for the parallel MI-MDS in Section 3.2. Parallel MI-MDS consists of two different computations, MI part and STRESS calculation part. MI part is pleasingly parallel and its computational complexity is $\mathcal{O}(M)$, where $M = N - n$, if the sample size n is considered as a constant. Since the MI 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 result to all the processes for the further computation. Thus, the communicated message amount through MPI primitives is $\mathcal{O}(M)$, 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 applied to the proposed symmetric pairwise computation in Section 3.3, each process uses `MPI_SENDRECV` k times to send 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 $\mathcal{O}(M/p \times p) = \mathcal{O}(M)$ without regard to the number of processes.

The MPI overhead during MI part and STRESS calculating part at Cluster-I and Cluster-II in Table 1 are shown in Fig. 8 and Fig. 9, 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 $\mathcal{O}(M)$ starting with the smallest sample size, here 12.5k. Both Fig. 8 and Fig. 9 show that the actual overhead measurement follows the MPI communication overhead model.

Fig. 10 and Fig. 11 illustrate the efficiency of Interpolation part and 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 following:

$$f = \frac{pT(p) - T(1)}{T(1)} \tag{30}$$

$$\varepsilon = \frac{1}{1+f} \tag{31}$$

where p is the number of parallel units, $T(p)$ is the running

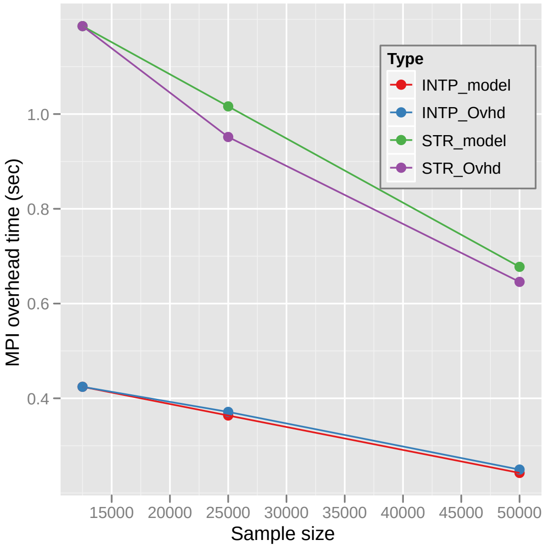


Figure 9: Parallel overhead modeled as due to MPI communication in terms of sample data size (m) using Cluster-II in Table 1 and message passing overhead model.

time with p parallel units, and $T(1)$ is the sequential running time. In practice, Eq. (30) can be replaced with following:

$$f = \frac{\alpha T(p_1) - T(p_2)}{T(p_2)} \tag{32}$$

where $\alpha = p_1/p_2$ and p_2 is the smallest number of used cores for the experiment, so $alpha \geq 1$. We use Eq. (32) for the overhead calculation.

In Fig. 10, 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. 11. Processes communicate via MPI primitives and each process is also parallelized in thread level. Both Fig. 10 and Fig. 11 show very good efficiency with appropriate degree of parallelism. Since both MI part and STRESS calcualtion part are pleasingly parallel within a process, the major overhead portion is the MPI message communicating overhead unless load balance is not achieved in thread-level parallelization within each process. In the previous paragraphs, the MPI communicating overhead is investigated and the MPI communicating overhead shows $\mathcal{O}(M)$ relation. Thus, the MPI overhead is constant if we examine with the same number of process and the same out-of-sample data size. Since the parallel computation time is decreased as more cores are used, but the overhead time is constant, it 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 environment, i.e. Giga bit ethernet and 20Gbps Infiniband. The difference is easily found by comparing Fig. 8 and Fig. 9.

5.2.2 Parallel GTM Interpolation

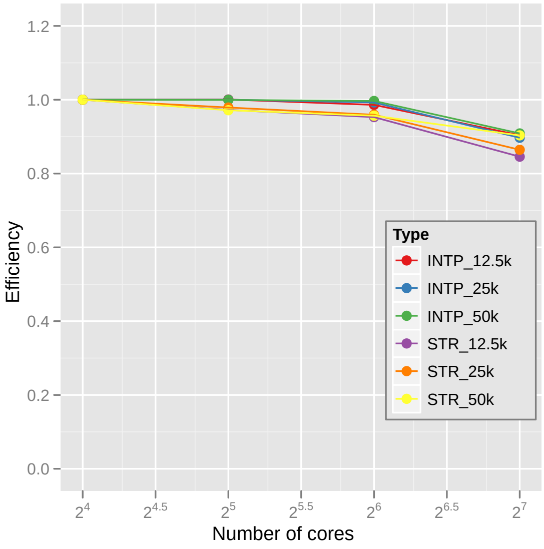


Figure 10: Efficiency of Interpolation part (INTP) and STRESS evaluation part (STR) runtime in parallel MI-MDS application with respect to different sample data size using Cluster-I in Table 1. Total data size is 100K.

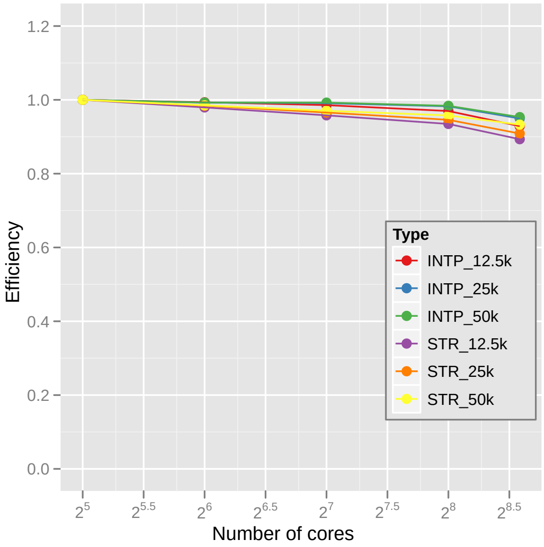


Figure 11: Efficiency of Interpolation part (INTP) and STRESS evaluation part (STR) runtime in parallel MI-MDS application with respect to different sample data size using Cluster-II in Table 1. Total data size is 100K.

We have measured performance of our parallel GTM interpolation algorithm discussed in 4.2 by using 16 nodes of the Cluster-II shown in Table 1, as we increase the number of cores from 16 to 256.

As shown in (29), the ideal efficiency of our parallel GTM interpolation algorithm is a constant 1.0 with respect to the number of PQ processes and can be degraded due to the net-

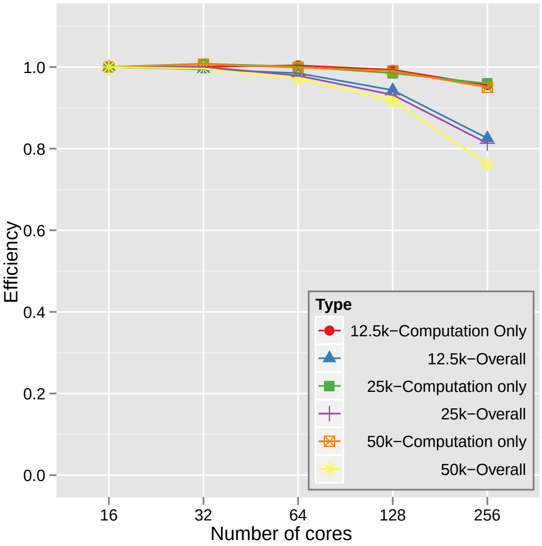


Figure 12: Efficiency of parallel GTM interpolation with respect to various sample sizes (12.5k, 25k, and 50k) and number of cores (from 32 up to 256 cores)

work latency which is mainly caused by the high number of communications. As we increase the number of cores (more decompositions), the communications between sub blocks are increasing and thus the network latency can affect the efficiencies.

As we expected, our experiment results shown in Fig. 12, the efficiency of our parallel GTM interpolation is close to 1 as we using the small number of cores but degraded as we have increased the number of processes(cores) for all data sizes (12.5k, 25k, and 50k) we used in our experiments. Also note that the efficiency for 12.5k dataset (meaning that, we have process 87.5k dataset for interpolation based on 12.5k random samples from the full 100k dataset and thus largest payload size in our test) is slightly better than one's from 50k dataset (meaning that, 50k data processed based on 50k samples of 100k dataset and thus smallest data size), as we increase the number of cores. This is also due to the network latency by which small data size is affected more.

With our parallel interpolation algorithms for MDS and GTM, we have also processed the large volume of PubChem data by using our Cluster-II and the results are shown in Fig. 13 and Fig. 14. We performed parallel MDS and GTM interpolation to process 100 thousands and 2 million dataset by using 100k PubChem data set as a training set. 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

We have proposed and tested interpolation algorithms for extending MDS and GTM dimension reduction approaches to very large datasets. We have shown that our interpolation approach gives results of good quality with high parallel performance. In quality comparison, the experimental results shows that the interpolation approach output is comparable to the normal MDS output which takes much more running time than interpolation. Furthermore, the interpolation approach enables us to visualize 2 million Pubchem data points in this paper and the size can be extended further with moderate running time. Future research includes application of these ideas to different areas including metagenomics and other DNA sequence visualization. We will research more detailed parallel performance for basic MDS and GTM algorithms in both traditional and deterministically annealed variations.

7. REFERENCES

[1] D. Aloise, A. Deshpande, P. Hansen, and P. Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, 2009.

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

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

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

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

[6] M. Carreira-Perpinán and Z. Lu. The laplacian eigenmaps latent variable model. In *Proc. of the 11th Int. Workshop on Artificial Intelligence and Statistics (AISTATS 2007)*. Citeseer, 2007.

[7] J. Y. Choi, S.-H. Bae, X. Qiu, and 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)*, May 2010.

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

[9] J. de Leeuw. Applications of convex analysis to multidimensional scaling. *Recent Developments in Statistics*, pages 133–145, 1977.

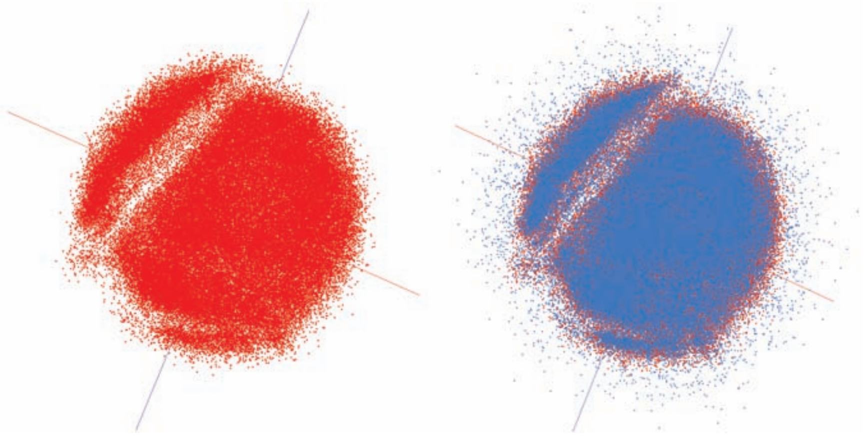
[10] J. de Leeuw. Convergence of the majorization method for multidimensional scaling. *Journal of Classification*, 5(2):163–180, 1988.

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

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

[13] G. Fox, S. Otto, and A. Hey. Matrix algorithms on a hypercube I: Matrix multiplication. *Parallel computing*, 4(1):17–31, 1987.

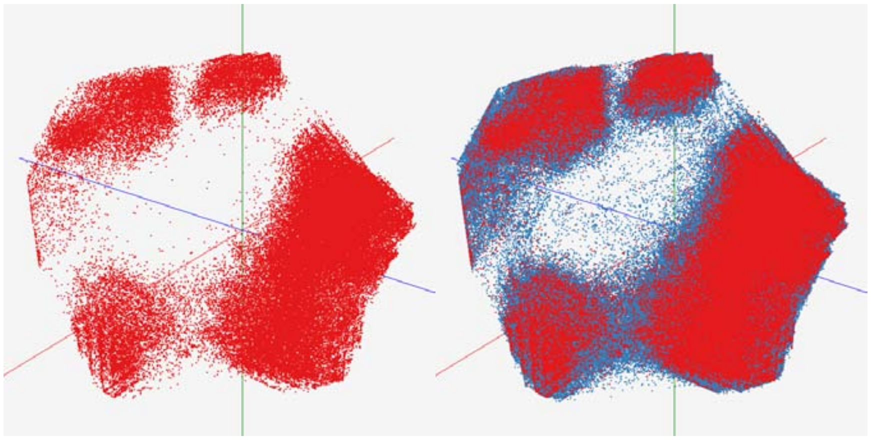
[14] S. Ingram, T. Munzner, and M. Olano. Glimmer: Multilevel mds on the gpu. *IEEE Transactions on*



(a) MDS 100k (trained set)

(b) MDS 200k

Figure 13: Interpolated MDS results. Based on 100k samples (a), additional 100k PubChem dataset is interpolated (b). Sampled data are colored in red and interpolated points are in blue.



(a) GTM 100k (trained set)

(b) GTM 2M

Figure 14: Interpolated GTM 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.

Visualization and Computer Graphics, 15(2):249–261, 2009.

[15] A. Kaban. A scalable generative topographic mapping for sparse data sequences. In *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05)-Volume I*, pages 51–56. Citeseer, 2005.

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

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

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

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

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

[21] M. W. Trosset and C. E. Priebe. The out-of-sample problem for classical multidimensional scaling. *Computational Statistics and Data Analysis*, 52(10):4635–4642, 2008.

[22] R. Van De Geijn and J. Watts. SUMMA: Scalable universal matrix multiplication algorithm. *Concurrency Practice and Experience*, 9(4):255–274, 1997.

[23] Z. Wang, S. Zheng, Y. Ye, and S. Boyd. Further relaxations of the semidefinite programming approach to sensor network localization. *SIAM Journal on Optimization*, 19(2):655–673, 2008.

[24] S. Xiang, F. Nie, Y. Song, C. Zhang, and C. Zhang. Embedding new data points for manifold learning via coordinate propagation. *Knowledge and Information Systems*, 19(2):159–184, 2009.