Scalable High Performance Dimension Reduction

Seung-Hee Bae
School of Informatics and Computing,
Indiana University, Bloomington IN, USA
{*sebae@indiana.edu*}

April 9, 2010

Contents

Abstract

Dimension reduction is a useful tool for visualization of such high-dimensional data to make data analysis feasible for such vast volume and high-dimensional scientific data. Among the known dimension reduction algorithms, multidimensional scaling algorithm is investigated in this proposal due to its theoretical robustness and high applicability. Multidimensional scaling is known as a non-linear optimization problem so that it is easy to be trapped in local optima if EM-like *hill-climbing* approach is used to solve it. In order to avoid local optima, the author has applied deterministic annealing (DA) approach to the well-known EM-like multidimensional scaling algorithm called SMACOF. Furthermore, the MDS algorithm is necessary to be parallelized to deal with large amount of data via distributed-memory environment, such as multicore cluster systems, since multidimensional scaling requires $\mathcal{O}(N^2)$ physical memory as well as $\mathcal{O}(N^2)$ computational resources. Although parallelization enables SMACOF algorithm to deal with tens of thousands or even hundreds of thousands points, it is still difficult to run parallel SMACOF algorithm with millions points, since it requires too much memory and computation to execute. Thus, the author proposes an interpolated approach to utilizing the known mapping of only a subset of the given data, named *in-sample* 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, the author presents how to parallelize proposed interpolation algorithms, as well. As we expected, the applying DA approach to SMACOF algorithm enables the proposed algorithm not to be stucked in the local optima but to find better results consistently with tested biological sequence data. Also, applying distributed parallelism to SMACOF algorithm helps to run with bigger data size which is not apt to a single compute node. The author is going to compare pure MPI parallel model with hybrid (MPI-Threading) parallel model to aim at finding better parallel model for the SMACOF (and DA-SMACOF) algorithm. Also, the experimental results illustrate that the quality of interpolated mapping results are comparable to the mapping results of original algorithm only. In parallel performance aspect, the interpolation method is parallelized with high efficiency. With the proposed interpolation method, it is possible to 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. The affect of the weight function in the STRESS value will also be investigating with several non-uniform weight function as well as uniform weight function.

1 Motivation

Due to the advancement of technology and sciences for last several decades, a huge amount of data are generated in every minute in the world from every technical and scientific fields. Nowadays are called data deluge era. In reflection of data deluge era, data-intensive scientific computing [10] has been emerging in the scientific computing fields and getting more interested by many people. Dealing with a huge amount of data is a major challenge in most of data mining and machine learning community. In addition, the invention of multicore chip has affected to the modern computing society since its invention. Now, parallelism is a key feature of software design and implementation for the high-performance computing. Due to the above facts, it is important to have high performance parallel data mining algorithms.

Most of the scientific data are high-dimensional data, so it is hard to understand data distribution in the original data space. Thus, it is beneficial to reduce their dimensionality to the visible dimension, i.e. two- or three-dimension. That is the reason why many dimension reduction algorithms have been proposed and applied to many applications. Many dimension reduction algorithms, such as principal component analysis (PCA), generative topographic mapping (GTM) [3, 4], and self-organizing map (SOM) [16], assume that the given data are presented in the form of vectors. However, for certain data, like biological sequences, it is hard to be represented by feature vectors though we could have some proximity information, such as distance or dissimilarity. Due to the above listed reasons, it is really interesting and useful to have high performance parallel dimension reduction algorithms, such as multidimensional scaling (MDS) [5,18], which build a configuration of the given data (points) in the target dimension based on proximity information, but not necessary to have feature vectors.

MDS algorithm is well-known as a non-linear optimization problem to configure a target dimensional mappings of high-dimensional data with respect to the given proximity information, whose Euclidean distance between two mapping points are similar to the corresponding proximity information as much as possible. Therefore, if we apply hill-climbing approach (or gradient descent approach) to MDS problem for finding optimal mapping, it will generally be kept at local optima. Since the local optima problem of hill-climbing approach is a typical issue of optimization problems, lots of alternative method has been proposed and used a lot in many applications. Here, deterministic annealing (DA) [22, 23] approach is one of the alternatives which shows two merits: (1) faster execution than stochastic approach, and (2) ability of escaping local optima.

In addition, MDS requires $\mathcal{O}(N^2)$ computation and memory, which becomes huge if we apply to large data set. Due to the memory resource limitation, it can be thought of as an *memory-bounded* application for large data set. For the purpose of dealing with large data set, it is essential to parallelize MDS algorithms in distributed memory model which results in utilizing distributed memory systems, such as cluster systems, to overcome out of memory problem.

Though we parallelize MDS application, it is impractical to run MDS with millions of points since the MDS algorithms requires memory $\mathcal{O}(N^2)$ as well as computation. This hinders to run even parallel MDS algorithm with millions of points due to out of memory. From that initiative, the interpolated approach of MDS is also proposed here to reduce the memory requirement dramatically so that it is possible to run MDS with even millions points with trade-off of a little quality deficience.

Based on the above motivations, the remainder of this paper is organized as follows. In Section 2, the overviews of the background methods are explained. Section 3 describes how to apply DA method to MDS problem based on SMACOF algorithm followed by explanation of how to apply *Out-of-Sample* approach to MDS in Section 4. The meaning of *non-uniform* weights are briefly introduced in Section 5 and parallelization is provided in Section 6. Section 7 illustrates the experimental results of those proposed methods followed by contribution of this study in Section 8.

2 Background and Related Work

2.1 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 that pursues a mapping in the target dimension whose pairwise distances are as similar as possible to the 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. Dissimilarity matrix $\boldsymbol{\Delta}$ has the following constraints: (1) Symmetric ($\delta_{ij} = \delta_{ji}$), (2) non-negative ($\delta_{ij} \geq 0$), and (3) zero diagonal ($\delta_{ii} = 0$). By MDS algorithm, the lower dimensional mapping is generated in target dimension, say L dimension, and the generated mapping could be also represented as an $N \times L$ matrix ($\boldsymbol{X}$), 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 by the well-known objective functions of MDS, namely STRESS [17] or SSTRESS [25]. Below equations are the definition of STRESS (1) and SSTRESS (2):

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

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

where $1 \leq i < j \leq N$ and w_{ij} is a weight value, so $w_{ij} \geq 0$. As STRESS and SSTRESS describe, the MDS problem is a non-linear optimization problem to find a mapping in target dimension which minimize the given objective function.

2.2 Scaling by MAjorizing a COmplicated Function (SMACOF)

Scaling by MAjorizing a COmplicated Function (SMACOF) [7, 8] is an iterative majorization algorithm to solve MDS problem with STRESS criterion. The iterative majorization procedure of the SMACOF is essentially the Expectation-Maximization (EM) [9] approach. Although SMACOF has a tendency to find local minima due to its hill-climbing attribute, it is still a powerful method since it is guaranteed to decrease STRESS (σ) criterion monotonically. Instead of mathematical detail explanation of SMACOF algorithm, the SMACOF procedure is shown in Alg. 1. The mathematical details of SMACOF algorithm is described well in [5].

Alg. 1 illustrates the SMACOF algorithm for MDS solution. The main procedure of SMACOF is iterative matrix multiplications, called Guttman transform, as shown at Line 9 in Alg. 1, where $\boldsymbol{V}^\dagger$ is the Moore-Penrose inverse [19, 20] (or pseudo-inverse) of matrix $\boldsymbol{V}$. The Guttman transform is induced by the stationary equation $\nabla\sigma(X) = 0$ which can be written as $VX = B(Z)Z$. The $N \times N$ matrices $\boldsymbol{V}$ and $\boldsymbol{B}(\boldsymbol{Z})$ are defined as follows:

Algorithm 1 SMACOF algorithm

```
Input: $\boldsymbol{\Delta}, \boldsymbol{V}^{\dagger}$
1: Generate random initial mapping $\boldsymbol{X}^{[0]}$.
2: $k \Leftarrow 0$;
3: $\varepsilon \Leftarrow$ small positive number;
4: $MAX \Leftarrow$ maximum iteration;
5: Compute $\sigma^{[0]} = \sigma(\boldsymbol{X}^{[0]})$;
6: while $k = 0$ or ($\Delta\sigma > \varepsilon$ and $k \leq MAX$) do
7:     $k \Leftarrow k + 1$;
8:     Update $\boldsymbol{B}(\boldsymbol{X}^{[k-1]})$.
9:     $\boldsymbol{X}^{[k]} = \boldsymbol{V}^{\dagger}\boldsymbol{B}(\boldsymbol{X}^{[k-1]})\boldsymbol{X}^{[k-1]}$
10:    Compute $\sigma^{[k]} = \sigma(\boldsymbol{X}^{[k]})$
11:    $\boldsymbol{Z} \Leftarrow \boldsymbol{X}^{[k]}$;
12: end while
13: return $\boldsymbol{Z}$;
```

$$\boldsymbol{V} = (v_{ij}) \tag{3}$$

$$v_{ij} = \begin{cases} -w_{ij} & \text{if } i \neq j \\ \sum_{i \neq j} w_{ij} & \text{if } i = j \end{cases} \tag{4}$$

$$\boldsymbol{B}(\boldsymbol{Z}) = (b_{ij}) \tag{5}$$

$$b_{ij} = \begin{cases} -w_{ij}\delta_{ij}/d_{ij}(\boldsymbol{Z}) & \text{if } d_{ij}(\boldsymbol{Z}) \neq 0, i \neq j \\ 0 & \text{if } d_{ij}(\boldsymbol{Z}) = 0, i \neq j \\ -\sum_{i \neq j} b_{ij} & \text{if } i = j \end{cases} \tag{6}$$

If the weights are equal to one ($w_{ij} = 1$) for all pairwise dissimilarity, then

$$\boldsymbol{V} = N\left(I - \frac{\boldsymbol{e}\boldsymbol{e}^t}{N}\right) \tag{7}$$

$$\boldsymbol{V}^{\dagger} = \frac{1}{N}\left(I - \frac{\boldsymbol{e}\boldsymbol{e}^t}{N}\right) \tag{8}$$

where $\boldsymbol{e} = (1, \ldots, 1)^t$ is one vector whose length is N.

As in Alg. 1, SMACOF algorithm requires $\mathcal{O}(N^2)$ computation, since Guttman transform performs multiplication of $N \times N$ matrix and $N \times L$ matrix twice, typically $N \gg L$, and computing STRESS value, $\boldsymbol{B}(\boldsymbol{X}^{[k]})$, and $\boldsymbol{D}(\boldsymbol{X}^{[k]})$ also take $\mathcal{O}(N^2)$. In addition, the SMACOF algorithm requires $\mathcal{O}(N^2)$ memory because it needs several $N \times N$ matrices as in Table 1. Therefore, it is impossible to run SMACOF for large data set under a typical single node computer due to the memory requirement increases in $\mathcal{O}(N^2)$. In order to overcome the memory shortage in a single node and to obtain computational benefit, it is essential to utilize distributed memory systems, i.e. cluster systems, with process level parallelism via message passing interface (MPI). It is illustrated in Section 6.1.

2.3 Deterministic Annealing Approach (DA)

SMACOF is a quite powerful algorithm, since it will monotonically decrease the STRESS criterion. However, the well-known problem of the gradient descent approach is to be kept easily by a local optima due to its hill-climbing property. Stochastic optimization approaches, such as simulated annealing (SA) [14] and genetic algorithms (GA) [11,13], have been used in order to avoid local optima, but those algorithms generally also suffer from using a huge amount of running time due to their random movement property.

Since the simulated annealing (SA) was introduced by Kirkpatrick et al. [14], people widely accepted SA and other stochastic maximum entropy approach to solve optimization problems for the purpose of finding global optimum instead of hill-climbing deterministic approaches. SA is a Metropolis algorithm, which accepts not only the better proposed solution than previous solution but even worse proposed solution than the previous solution based on a certain probability which is related to *computational temperature* (T). Also, it is known that Metropolis algorithm converges to an equilibrium probability distribution known as *Gibbs probability distribution.* If we denote $H(\boldsymbol{X})$ as the energy (or cost) function and $\mathcal{F}$ as a *free energy*, then Gibbs distribution density is following:

$$P^G(\boldsymbol{X}) = \exp\left(-\frac{1}{T}(\boldsymbol{H}(\boldsymbol{X}) - \mathcal{F})\right), \tag{9}$$

$$\mathcal{F} = -T \log \int \exp\left(-\frac{1}{T}\boldsymbol{H}(\boldsymbol{X})\right) \mathrm{d}\boldsymbol{X}. \tag{10}$$

and the *free energy ($\mathcal{F}$)*, which is an objective function, is minimized by the Gibbs probability density P^G. Also, free energy can be written as following:

$$\mathcal{F}_P = < \boldsymbol{H} >_P - T\mathcal{S}(P) \tag{11}$$

where $< \boldsymbol{H} >_P$ represents the *expected energy* and $\mathcal{S}(P)$ is *entropy* of the system with probability density P. Here, T is used as a Lagrange multiplier to control the expected energy. Eq. (11) illustrates the analogy of physical annealing of solids, which solutions for the optimization problem are affected by the computational temperature (T). With high temperature, the problem space is dominated by the *entropy* term which make the problem space become smooth. As temperatures is getting cooler, however, the problem space is gradually revealed as the landscape of the original cost functions. To avoid trapped in local optima, people usually start with high temperature and slowly decrease temperature in the process of finding solution.

SA relies on random sampling with Monte Carlo method to estimate the expected solution, e.g. expected mapping in target dimension for MDS problem, so that it suffers from long running time. In contrast, deterministic annealing (DA) [22, 23] method actually tries to calculate the expected solution exactly or approximately with respect to the Gibbs distribution as an amendment of SA's long running time, while it follows computational annealing process using Eq. (11), which T decreases from high to low.

DA method is used for many optimization problems, including clustering [22, 23], pairwise clustering [12], and MDS [15]. Since it is intractable to calculate $\mathcal{F}$ in Eq. (10) exactly, an approximation technique called *mean field* approximation is used for solving MDS problem by DA in [15], in that Gibbs distribution $P^G(\boldsymbol{X})$ is approximated by a factorized distribution with density

$$P^0(\boldsymbol{X}|\boldsymbol{\Theta}) = \prod_{i=1}^{N} q_i(\boldsymbol{x}_i|\boldsymbol{\Theta}_i). \tag{12}$$

where $\boldsymbol{\Theta}_i$ is a vector of mean field parameter of $\boldsymbol{x}_i$ and $q_i(\boldsymbol{x}_i|\boldsymbol{\Theta}_i)$ is a factor serves as a marginal distribution model of the coordinates of $\boldsymbol{x}_i$. To optimize parameters $\boldsymbol{\Theta}_i$, Klock and Buhmann [15] minimized Kullback-Leibler (KL) divergence between the approximated density $P^0(\boldsymbol{X})$ and the Gibbs density $P^G(\boldsymbol{X})$ through EM algorithm [9]. The alternatively proposed DA-MDS algorithm is described at Section 3.

2.4 ***Out-of-Sample*** Problems

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 [29] provides a generalized out-of-sample solutions for non-linear dimension reduction problems by using coodinate propagation. In sensor network localization field, one of the interesting problem is how to find out the locations of the remaining sensors, when there are only a subset of pairwise distances between sensors and a subset of anchor locations are available. For that problem, semi-definite programming relaxation approaches and its extended approaches has been proposed to solve it by Wang et. al. in [28].

Also, *Out-of-sample* approach have been investigated in MDS research community by several people, specially based on classical multidimensional scaling (CMDS) [26]. CMDS [26] generates the embeddings in the configured space based on spectral decomposition of a symmetric positive semidefinite matrix (or the approximation of positive semidefinite matrix), and the out-of-sample extensions of CMDS are proposed in [2, 27]. [2] projected the new point $\boldsymbol{x}$ onto the principal components, and [27] extends the CMDS algorithm itself to the out-of-sample problem. Trosset and Priebe proposed a new embedding mechanism of a new point with respect to the pre-mapped configurations of the sampled n objects by modifying the original CMDS equations in [27], in that it preserves the mappings of the original n objects based on $(n+1)\times(n+1)$ matrix $\boldsymbol{A}_2$ instead of $n \times n$ matrix $\boldsymbol{\Delta}_2$, and extends to embedding a number of points simultaneously.

In contrast to applying out-of-sample problem to CMDS, we extend 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 majorization. The proposed iterative majorizing interpolation approach for the MDS problem will be explained in Section 4.

3 Deterministic Annealing SMACOF

If we use STRESS (1) objective function as an expected energy (cost) function in Eq. (11), then we can define $\mathcal{H}_{MDS}$ and $\mathcal{H}_0$ as following:

$$\mathcal{H}_{MDS} = \sum_{i<j\leq N}^{N} w_{ij}(d_{ij}(\boldsymbol{X}) - \delta_{ij})^2 \tag{13}$$

$$\mathcal{H}_0 = \sum_{i=1}^{N} \frac{(\boldsymbol{x}_i - \boldsymbol{\mu}_i)^2}{2} \tag{14}$$

where $\mathcal{H}_0$ corresponds to an energy function based on a simple multivariate Gaussian distribution and $\boldsymbol{\mu}_i$ represents the average of the multivariate Gaussian distribution of i-th point ($i = 1, \ldots, N$) in target dimension (L-dimension). Also, we define P^0 and $\mathcal{F}_0$ as following:

$$P^0(\boldsymbol{X}) = \exp\left(-\frac{1}{T}(\mathcal{H}_0 - \mathcal{F}_0)\right), \tag{15}$$

$$\mathcal{F}_0 = -T \log \int \exp\left(-\frac{1}{T}\mathcal{H}_0\right) \mathrm{d}\boldsymbol{X} = -T \log (2\pi T)^{L/2} \tag{16}$$

We need to minimize $\mathcal{F}_{MDS}(P^0) =< \mathcal{H}_{MDS} - \mathcal{H}_0 > +\mathcal{F}_0(P^0)$ with respect to $\boldsymbol{\mu}_i$. Since $- < \mathcal{H}_0 > +\mathcal{F}_0(P^0)$ is independent to $\boldsymbol{\mu}_i$, only $< \mathcal{H}_{MDS} >$ part is necessary to be minimized with regard to $\boldsymbol{\mu}_i$. If we apply $< \boldsymbol{x}_i\boldsymbol{x}_i >= \boldsymbol{\mu}_i\boldsymbol{\mu}_i + TL$ to $< \mathcal{H}_{MDS} >$, then $< \mathcal{H}_{MDS} >$ can be deployed as following:

$$< \mathcal{H}_{MDS} > = \sum_{i<j\leq N}^{N} w_{ij}(< \|\boldsymbol{x}_i - \boldsymbol{x}_j\| > -\delta_{ij})^2 \tag{17}$$

$$\approx \sum_{i<j\leq N}^{N} w_{ij}(\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\| + \sqrt{2TL} - \delta_{ij})^2 \tag{18}$$

where $\|\boldsymbol{a}\|$ is Norm$_2$ of a vector $\boldsymbol{a}$. Eq. (17) can be approximated to Eq. (18), since the bigger T, the smaller $\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|$ and vice versa.

In [15], Klock and Buhmann tried to find an approximation of $P^G(\boldsymbol{X})$ with mean field factorization method by minimizing Kullback-Leibler (KL) divergence using EM approach. The found parameters by minimizing KL-divergence between $P^G(\boldsymbol{X})$ and $P^0(\boldsymbol{X})$ using EM approach are essentially the expected mapping in target dimension under current problem space with computational temperature (T).

In contrast, we try to find expected mapping, which minimize $\mathcal{F}_{MDS}(P^0)$, directly with new objective function ($\hat{\sigma}$) which is applied DA approach to MDS problem space with computational temperature T by well-known EM-like MDS solution, called SMACOF [7, 8]. Therefore, as T varies, the problem space also varies, and SMACOF algorithm is used to find expected mapping under each problem space at a corresponding T. In order to apply DA method to SMACOF algorithm, we substitute the original STRESS equation (1) with Eq. (18). Note that $\boldsymbol{\mu}_i$ and $\boldsymbol{\mu}_j$ are the expected mappings we are looking for, so we can consider $\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|$ as $d_{ij}(\boldsymbol{X}_T)$, where $\boldsymbol{X}_T$ represents the embedding results in L-dimension at T and d_{ij} means the Euclidean distance between mappings of point i and j. Thus, the new STRESS ($\hat{\sigma}$) is following:

$$\hat{\sigma} = \sum_{i<j\leq N}^{N} w_{ij}(d_{ij}(\boldsymbol{X}_T) + \sqrt{2TL} - \delta_{ij})^2 \tag{19}$$

$$= \sum_{i<j\leq N}^{N} w_{ij}(d_{ij}(\boldsymbol{X}_T) - \hat{\delta}_{ij})^2 \tag{20}$$

with defined $\hat{\delta}_{ij}$ as following:

$$\hat{\delta}_{ij} = \begin{cases} \delta_{ij} - \sqrt{2TL} & \text{if } \delta_{ij} > \sqrt{2TL} \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

In addition, T is a lagrange multiplier so it can be thought of as $T = \hat{T}^2$, then $\sqrt{2TL} = \hat{T}\sqrt{2L}$ and we will use T instead of $\hat{T}$ for the simple notation. Thus, Eq. (21) can be written as following:

$$\hat{\delta}_{ij} = \begin{cases} \delta_{ij} - T\sqrt{2L} & \text{if } \delta_{ij} > T\sqrt{2L} \\ 0 & \text{otherwise.} \end{cases} \tag{22}$$

Now, we can apply DA approach to SMACOF algorithm by adapting the problem space based on computational temperature T via new STRESS (20). The MDS problem space could be smoother with higher T than with lower T, since T represents the portion of entropy to the free energy $\mathcal{F}$ as in Eq. (11). Generally, DA approach starts with high T and gets cool down T as time goes on, like physical annealing process. However, if starting computational temperature (T_0) is very high which results in all $\hat{\delta}_{ij}$ become ZERO, then all points will be mapped at origin (O). Once all mappings are at the origin, then the Guttman transform which is shown at Line 9 in Alg. 1 is unable to construct other mapping except the mapping of all at the origin, since Guttman transform does multiplication iteratively with previous mapping to calculate current mapping. Thus, we need to calculate T_0 which makes at least one $\hat{\delta}_{ij}$ is bigger than ZERO, so that at least one of the points is not located at O. If δ_{max} denotes the $\mathbf{max}(\delta_{ij})$, where $0 < i < j \leq N$, then we can compute T_0 like:

$$T_0 = \frac{\alpha}{\sqrt{2L}}\delta_{max} \tag{23}$$

where α ($0 < \alpha < 1$) is the cooling parameter to reduce computational temperature, as DA method proceeds. With computed T_0, the $\widehat{\boldsymbol{\Delta}}_0 = [\hat{\delta}_{ij}]$ can be calculated, and we are able to run SMACOF algorithm with respect to Eq. (20). After new mapping generated with T_0 by SMACOF algorithm, say $\boldsymbol{X}_0$, then we will cool down the temperature in exponential way, like $T_1 = \alpha T_0$, and keep doing above steps until T becomes too small. Finally, we set $T = 0$ and then run SMACOF by using the latest mapping as an initial mapping with respect to original

Algorithm 2 DA-SMACOF algorithm

```
Input: $\boldsymbol{\Delta}, \boldsymbol{V}^{\dagger}$ and $\alpha$
 1: Compute $T_0$ in Eq. (23).
 2: Compute $\widehat{\boldsymbol{\Delta}}_0$ with $T_0$ based on Eq. (22)
 3: Generate random initial mapping $\boldsymbol{X}_0$.
 4: $k \Leftarrow 0$;
 5: while $T_k \geq T_{min}$ do
 6:     $\boldsymbol{X}_{k+1}$ = output of SMACOF in Alg. 1 with $\widehat{\boldsymbol{\Delta}}_k, \boldsymbol{V}^{\dagger}$ and $\boldsymbol{X}_k$. $\boldsymbol{X}_k$ is used for initial
        mapping of the current SMACOF running.
 7:     Cool down computational Temperature $T_{k+1} = \alpha T_k$
 8:     Update $\widehat{\boldsymbol{\Delta}}_{k+1}$ w.r.t. $T_{k+1}$.
 9:     $k \Leftarrow k + 1$;
10: end while
    /* Finally, we will run SMACOF with original dissimilarity matrix ($\boldsymbol{\Delta}$) by using $\boldsymbol{X}_k$ as the
    initial mapping. */
11: $\boldsymbol{X}$ = output of SMACOF based on $\boldsymbol{\Delta}, \boldsymbol{V}^{\dagger}$ and $\boldsymbol{X}_k$.
12: return: $\boldsymbol{X}$;
```

STRESS (1). We will assume the uniform weight $\forall w_{ij} = 1$ where $0 < i < j \leq N$, so that Eq. (8) is used to SMACOF running. The proposed deterministic annealing SMACOF algorithm, called DA-SMACOF is illustrated in Alg. 2.

4 Majorizing Interpolation MDS (MI-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) [7, 8], a well-known MDS application via Expectation-Maximization (EM) [9] 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 2 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 2 (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, the author developed a simple interpolation approach based on pre-mapped MDS result of the sample of the given data. The suggested interpolation algorithm is similar to k nearest neighbor (k-NN) classification [6], but it aims to find 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 new mapping position in relation to the k-NN positions, iterative majorization method is used as in SMACOF [7, 8] 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 ($\mathbf{\Delta} = [\delta_{ij}]$) of those data as in Section 2.1. 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 $\boldsymbol{X}$, 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 $k+1$ points in $\boldsymbol{S}$, where $\boldsymbol{S} = \boldsymbol{P} \cup \boldsymbol{x}$. However, only one point $\boldsymbol{x}$ is movable among $k+1$ points, so we can summarize STRESS (1) as belows, and we set $w_{ij} = 1$, for $\forall i, j$ in order to simplify.

$$\sigma(\boldsymbol{S}) = \sum_{i<j\leq k+1} (d_{ij}(\boldsymbol{S}) - \delta_{ij})^2 \tag{24}$$

$$= \mathcal{C} + \sum_{i=1}^{k} d_{ix}^2 - 2\sum_{i=1}^{k} \delta_{ix} d_{ix} \tag{25}$$

where δ_{ix} is the original dissimilarity value between $\boldsymbol{p}_i$ and $\boldsymbol{x}$, 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. (25) can be deployed as following:

$$\sum_{i=1}^{k} d_{ix}^2 = k\|\boldsymbol{x}\|^2 + \sum_{i=1}^{k} \|\boldsymbol{p}_i\|^2 - 2\boldsymbol{x}^t\boldsymbol{q} \tag{26}$$

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 means the number of *nearest neighbors*. In order to establish majorizing inequality, we apply *Cauchy-Schwarz* inequality to $-d_{ix}$ of the third term of Eq. (25). 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:

$$-d_{ix} \leq \frac{(\boldsymbol{p}_i - \boldsymbol{x})^t(\boldsymbol{p}_i - \boldsymbol{z})}{d_{iz}} \tag{27}$$

where $\boldsymbol{z}^t = (z_i, \ldots, z_L)$ and $d_{iz} = \|\boldsymbol{p}_i - \boldsymbol{z}\|$. The equality in Eq. (27) occurs if $\boldsymbol{x}$ and $\boldsymbol{z}$ are equal. If we apply Eq. (27) to the third term of Eq. (25), then we obtain

$$-\sum_{i=1}^{k} \delta_{ix} d_{ix} \leq -\boldsymbol{x}^t \sum_{i=1}^{k} \frac{\delta_{ix}}{d_{iz}}(\boldsymbol{z} - \boldsymbol{p}_i) + \mathcal{C}_\rho \tag{28}$$

where $\mathcal{C}_\rho$ is a constant. If Eq. (26) and Eq. (28) are applied to Eq. (25), then it could be like following:

$$\begin{aligned}
\sigma(\boldsymbol{S}) &= \mathcal{C} + \sum_{i=1}^{k} d_{ix}^2 - 2\sum_{i=1}^{k} \delta_{ix} d_{ix} && (29)\\
&\leq \mathcal{C} + k\|\boldsymbol{x}\|^2 - 2\boldsymbol{x}^t\boldsymbol{q} + \sum_{i=1}^{k}\|\boldsymbol{p}_i\|^2 - \boldsymbol{x}^t \sum_{i=1}^{k} \frac{\delta_{ix}}{d_{iz}}(\boldsymbol{z} - \boldsymbol{p}_i) + \mathcal{C}_\rho && (30)\\
&= \tau(\boldsymbol{x}, \boldsymbol{z}) && (31)
\end{aligned}$$

where both $\mathcal{C}$ and $\mathcal{C}_\rho$ are constants. In the Eq. (31), $\tau(\boldsymbol{x}, \boldsymbol{z})$, 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

$$\begin{aligned}
\nabla\tau(\boldsymbol{x}, \boldsymbol{z}) &= 2k\boldsymbol{x} - 2\boldsymbol{q} - 2\sum_{i=1}^{k} \frac{\delta_{ix}}{d_{iz}}(\boldsymbol{z} - \boldsymbol{p}_i) = 0 && (32)\\
\boldsymbol{x} &= \frac{\boldsymbol{q} + \sum_{i=1}^{k} \frac{\delta_{ix}}{d_{iz}}(\boldsymbol{z} - \boldsymbol{p}_i)}{k}. && (33)
\end{aligned}$$

The advantage of the iterative majorization algorithm is that it guarantees to produce 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. (33), then we generate an iterative majorizing equation like following:

$$\boldsymbol{x}^{[t]} = \overline{\boldsymbol{p}} + \frac{1}{k}\sum_{i=1}^{k} \frac{\delta_{ix}}{d_{iz}}(\boldsymbol{x}^{[t-1]} - \boldsymbol{p}_i) \tag{34}$$

where $d_{iz} = \|\boldsymbol{p}_i - \boldsymbol{x}^{[t-1]}\|$ and $\overline{\boldsymbol{p}}$ is the average of k-NN's mapping results. Eq. (34) 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

$$\Delta\sigma(\boldsymbol{S}^{[t]}) = \sigma(\boldsymbol{S}^{[t-1]}) - \sigma(\boldsymbol{S}^{[t]}) < \varepsilon, \tag{35}$$

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 sample data is described in Alg. 3. Note that the algorithm uses

Algorithm 3 Majorizing Interpolation (MI) algorithm

```
1: 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 $\delta_{ix}$.
2: Gather mapping results in target dimension of the $k$-NN.
3: Calculate $\overline{\boldsymbol{p}}$, the average of pre-mapped results of $\boldsymbol{p}_i \in \boldsymbol{P}$.
4: Generate initial mapping of $\boldsymbol{x}$, called $\boldsymbol{x}^{[0]}$, either $\overline{\boldsymbol{p}}$ or a random point.
5: Compute $\sigma(\boldsymbol{S}^{[0]})$, where $\boldsymbol{S}^{[0]} = \boldsymbol{P} \cup \{\boldsymbol{x}^{[0]}\}$.
6: while $t = 0$ or $(\Delta\sigma(\boldsymbol{S}^{[t]}) > \varepsilon$ and $t \leq$ MAX_ITER) do
7:    increase $t$ by one.
8:    Compute $\boldsymbol{x}^{[t]}$ by Eq. (34).
9:    Compute $\sigma(\boldsymbol{S}^{[t]})$.
10: end while
11: return $\boldsymbol{x}^{[t]}$;
```

$\overline{\boldsymbol{p}}$ 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 $\boldsymbol{x}^{[0]}$.

5 Non-uniform Weights

The STRESS equation (1) could be considered as the sum of weighted squared error. Due to the simplicity and easy computation, *uniform* weight (where $\forall w_i j = 1, 1 \leq i, j \leq N$) is assumed in many case. However, the MDS mapping result can be quite different with respect to weight function, as in [15, 24]. Generally, *uniform* weight method will do more focus on larger distance entries than smaller distances, but *reciprocal* weight method ($w_{ij} = 1/\delta_{ij}$) gives more weights on the shorter distance. However, it is hard to say which weight function is generally better than the other. It depends on the purpose of the finding embeddings of the given data in target dimension. In this dissertation, the author will investigate various weight function, such as $1/\delta^2$ and $1/\sqrt{\delta}$ as well as *uniform* and *reciprocal* weight which is used for Sammon's mapping [24].

6 Parallel MDS algorithms

We have observed that processing very large dataset with a data mining algorithm is no more *cpu-bounded* computation but rather it is *memory-bounded* in that memory consumption is beyond the ability of a single process or even a single machine. Thus, running machine learning algorithms to process large dataset, including MDS, in a distributed fashion is crucial so that we can utilize multiple processes and distributed resources to handle very large data which usually not fit in the memory of a single process or a compute node. The problem becomes more obvious if the running OS is 32-bit which can handle at most 4GB virtual memory per process. To process large data with efficiency, parallel version of MDS has been developed by using Message Passing Interface (MPI) fashion. Also, the author is studying hybrid (MPI-Threading)

Table 1: Main matrices used in SMACOF

Matrix	Size	Description
$\boldsymbol{\Delta}$	$N \times N$	Matrix for the given pairwise dissimilarity $[\delta_{ij}]$
$\boldsymbol{D}(\boldsymbol{X})$	$N \times N$	Matrix for the pairwise Euclidean distance of mapped in target dimension $[d_{ij}]$
$\boldsymbol{V}$	$N \times N$	Matrix defined the value v_{ij} in (3)
$\boldsymbol{V}^{\dagger}$	$N \times N$	Matrix for pseudo-inverse of $\boldsymbol{V}$
$\boldsymbol{B}(\boldsymbol{Z})$	$N \times N$	Matrix defined the value b_{ij} in (5)
$\boldsymbol{W}$	$N \times N$	Matrix for the weight of the dissimilarity $[w_{ij}]$
$\boldsymbol{X}^{[k]}$	$N \times L$	Matrix for current L-dimensional configuration of N data points $\boldsymbol{x}_i^{[k]} (i = 1, \ldots, N)$
$\boldsymbol{X}^{[k-1]}$	$N \times L$	Matrix for previous L-dimensional configuration of N data points $\boldsymbol{x}_i^{[k-1]} (i = 1, \ldots, N)$

parallel model MDS.

6.1 Message Passing Model SMACOF

Table 1 describes frequently used matrices in SMACOF algorithm, and memory requirement of SMACOF algorithm increases quadratically as N increases. For the small dataset, memory would not be any problem. However, it turns out to be critical problem when we deal with large data set, such as thousands or even millions. For instance, if $N = 10,000$, then one $N \times N$ matrix of 8-byte double-precision numbers consumes 800 MB of main memory, and if $N = 100,000$, then one $N \times N$ matrix uses 80 GB of main memory. To make matters worse, SMACOF algorithm generally needs six $N \times N$ matrices, so at least 480 GB of memory is required to run SMACOF with 100,000 data points without considering two $N \times L$ configuration matrices in Table 1. If the weight is uniform ($w_{ij} = 1, \forall i, j$), we can use only four constants for representing $N \times N$ $\boldsymbol{V}$ and $\boldsymbol{V}^{\dagger}$ matrices in order to saving memory space. We, however, still need at least three $N \times N$ matrices, i.e. $\boldsymbol{D}(\boldsymbol{X})$, $\boldsymbol{\Delta}$, and $\boldsymbol{B}(\boldsymbol{X})$, which requires 240 GB memory for the above case, which is still infeasible amount of memory for a typical computer. That is why we have to implement parallel version of SMACOF with MPI.

To parallelize SMACOF, it is essential to ensure load balanced data decomposition as much as possible. Load balance is important not only for distribution of memory but also for distribution of computation, since parallelization makes implicit benefit to computation as well as memory distribution, due to less computing per process. One simple approach of data decomposition is that we assume $p = n^2$, where p is the number of processes and n is an integer. Though it is relatively less complicated decomposition than others, one major problem of this approach is that it is a quite strict constraint to utilize available computing processors (or cores). In order to release that constraint, we decompose an $N \times N$ matrix to $m \times n$ block decomposition, where m is the number of block rows and n is the number of block columns, and the only constraint of the decomposition is $m \times n = p$, where $1 \leq m, n \leq p$. Thus, each process requires only approximately $1/p$ of full memory requirements of SMACOF algorithm. The matrix $\boldsymbol{M}$ in Fig. 1 illustrates how we decompose each $N \times N$ matrices with 6 processes and $m = 2, n = 3$. Without loss of generality, we assume $N\%m = N\%n = 0$ in Fig. 1.

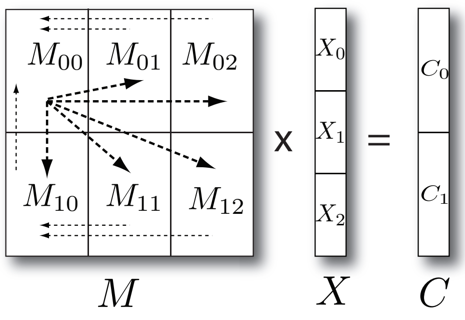


Figure 1: Parallel matrix multiplication of $N \times N$ matrix and $N \times L$ matrix with 6 processes and $m = 2, n = 3$.

A process $P_k, 0 \leq k < p$ (sometimes, we will use P_{ij} for matching $\boldsymbol{M}_{ij}$) is assigned to one rectangular block $\boldsymbol{M}_{ij}$ with respect to simple block assignment equation in (36):

$$k = i \times n + j \tag{36}$$

where $0 \leq i < m, 0 \leq j < n$. For $N \times N$ matrices, such as $\boldsymbol{\Delta}, \boldsymbol{V}^{\dagger}, \boldsymbol{B}(\boldsymbol{X}^{[k]})$, and so on, each block $\boldsymbol{M}_{ij}$ is assigned to the corresponding process P_{ij}, and for $\boldsymbol{X^{[k]}}$ and $\boldsymbol{X^{[k-1]}}$ matrices, $N \times L$ matrices, each process has full $N \times L$ matrices because these matrices are relatively much small size and it results in reducing a number of additional message passing. By scattering decomposed blocks to distributed memory, now we are able to run SMACOF with huge data set as much as distributed memory allows in the cost of message passing overheads and complicated implementation.

At the iteration k in Alg. 1, the application should be possible to acquire following information to do Line 9 and Line 10 in Alg. 1: $\boldsymbol{\Delta}$, $\boldsymbol{V}^{\dagger}$, $\boldsymbol{B}(\boldsymbol{X}^{[k-1]})$, $\boldsymbol{X}^{[k-1]}$, and $\sigma^{[k]}$. One good feature of SMACOF algorithm is that some of matrices are invariable, i.e. $\boldsymbol{\Delta}$ and $\boldsymbol{V}^{\dagger}$, through the iterations. On the other hand, $\boldsymbol{B}(\boldsymbol{X}^{[k-1]})$ and STRESS ($\sigma^{[k]}$) value keep changing at each iteration, since $d_{ij}(\boldsymbol{X}^{[k]})$ varies every iteration. In addition, in order to update $\boldsymbol{B}(\boldsymbol{X}^{[k-1]})$ and STRESS ($\sigma^{[k]}$) value in each iteration, we have to take $N \times N$ matrices information into account, so related processes should communicate via MPI primitives to obtain necessary information. Therefore, it is necessary to design message passing schemes to do parallelization for calculating $\boldsymbol{B}(\boldsymbol{X}^{[k-1]})$ and STRESS ($\sigma^{[k]}$) value as well as parallel matrix multiplication in Line 9 in Alg. 1.

Computing STRESS in (1) can be implemented simply through `MPI_Allreduce`. On the other hand, calculation of $\boldsymbol{B}(\boldsymbol{X}^{[k-1]})$ and parallel matrix multiplication is not simple, specially for the case of $m \neq n$. Fig. 1 depicts how parallel matrix multiplication applies between an $N \times N$ matrix $\boldsymbol{M}$ and an $N \times L$ matrix $\boldsymbol{X}$. Parallel matrix multiplication for SMACOF algorithm is implemented in three-step of message communication via MPI primitives. Block matrix multiplication of Fig. 1 for acquiring $\boldsymbol{C}_i$ $(i = 0, 1)$ can be written as follows:

$$\boldsymbol{C}_i = \sum_{0 \leq j < 3} \boldsymbol{M}_{ij} \cdot \boldsymbol{X}_j \tag{37}$$

Since $\boldsymbol{M}_{ij}$ of $N \times N$ matrix is accessed only by the corresponding process P_{ij}, computing $\boldsymbol{M}_{ij} \cdot \boldsymbol{X}_j$

Algorithm 4 Pseudo-code for distributed parallel matrix multiplication in SMACOF algorithm

```
Input: $\boldsymbol{M}_{ij}, \boldsymbol{X}$
 1: /* $m$ = Row Blocks, $n$ = Column Blocks */
 2: /* $i$ = Rank-In-Row, $j$ = Rank-In-Column */
 3: $\boldsymbol{T}_{ij} = \boldsymbol{M}_{ij} \cdot \boldsymbol{X}_j$
 4: if $j \neq 0$ then
 5:    Send $\boldsymbol{T}_{ij}$ to $P_{i0}$
 6: else
 7:    for $j = 1$ to $n - 1$ do
 8:       Receive $\boldsymbol{T}_{ij}$ from $P_{ij}$
 9:    end for
10:    Generate $\boldsymbol{C}_i$
11: end if
12: if $i == 0$ and $j == 0$ then
13:    for $i = 1$ to $m - 1$ do
14:       Receive $\boldsymbol{C}_i$ from $P_{i0}$
15:    end for
16:    Combine $\boldsymbol{C}$ with $\boldsymbol{C}_i$ where $i = 0, \ldots, m - 1$
17:    Broadcast $\boldsymbol{C}$ to all processes
18: else if $j == 0$ then
19:    Send $\boldsymbol{C}_i$ to $P_{00}$
20:    Receive Broadcasted $\boldsymbol{C}$
21: else
22:    Receive Broadcasted $\boldsymbol{C}$
23: end if
```

part is done by P_{ij}, and the each computed sub-matrix, which is $\frac{N}{2} \times L$ matrix for Fig. 1, is sent to the process assigned $\boldsymbol{M}_{i0}$ by MPI primitives, such as `MPI_Send` and `MPI_Receive`. Then the process assigned $\boldsymbol{M}_{i0}$, say P_{i0}, sums the received sub-matrices to generate $\boldsymbol{C}_i$, and send $\boldsymbol{C}_i$ block to P_{00}. Finally, P_{00} combines sub-matrix block $\boldsymbol{C}_i$ $(0 \leq i < m)$ to construct $N \times L$ matrix $\boldsymbol{C}$, and broadcast it to all other processes by `MPI_Broadcast`. Each arrows in Fig. 1 represents message passing direction. Thin dashed arrow lines describes message passing of $\frac{N}{2} \times L$ sub-matrices by `MPI_Send` and `MPI_Receive`, and message passing of matrix $\boldsymbol{C}$ by `MPI_Broadcast` is represented by thick dashed arrow lines. The pseudo code for parallel matrix multiplication in SMACOF algorithm is in Alg. 4

For the purpose of parallel computing $\boldsymbol{B}(\boldsymbol{X}^{[k-1]})$, whose elements b_{ij} is defined in (6), message passing mechanism in Fig. 2 should be applied under 2×3 block decomposition as in Fig. 1. Since $b_{ss} = -\sum_{s \neq j} b_{sj}$, a process P_{ij} who is assigned to $\boldsymbol{B}_{ij}$ should communicate a vector $\boldsymbol{s}_{ij}$, whose element is the sum of corresponding rows, with processes assigned sub-matrix of the same block-row P_{ik}, where $k = 0, \ldots, n - 1$, unless the number of column blocks is 1 $(n == 1)$. In Fig. 2, the diagonal dashed line indicates the diagonal elements, and the green colored blocks are diagonal blocks for each block-row. Note that the definition of *diagonal blocks*

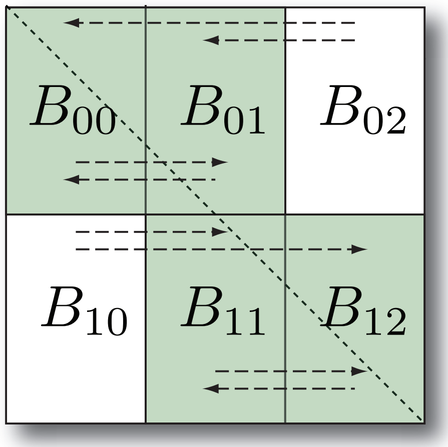


Figure 2: Calculation of $\boldsymbol{B}(\boldsymbol{X}^{[k-1]})$ matrix with regard to the decomposition of Fig. 1.

is a block which contains at least one diagonal element of the matrix $\boldsymbol{B}(\boldsymbol{X}^{[k]})$. Also, dashed arrow lines illustrate message passing direction.

6.2 Hybrid Model SMACOF

In the above, the author discussed the pure MPI model SMACOF implementation. The MPI model has been widely accepted by parallel computing community in order to utilize distributed memory computing resources, such as cluster, since it enables us to run such programs that require too large computation and memory to run in a single node. Though MPI model is still used very much and shows good performance, you should note that the model proposed several decades ago, when the increase of clock speed is the main method of improving CPU performance. After Multicore chip was invented, most of the cluster systems are actually composed of multicore compute nodes. For instance, both **Cluster-I** and **Cluster-II** in Table 2 are multicore cluster systems whose compute nodes contain 16 and 24 cores, correspondingly.

In [10], the authors investigated the overhead of pure MPI and hybrid (MPI-Threading) model with multicore cluster systems. In the paper, pure MPI outperforms hybrid model for the application with relatively fast message passing synchronization overhead. However, for the case of high MPI synchronization time, hybrid model outperforms pure MPI model with high parallelism. Not only collective MPI operations but also pairwise MPI operations, such as `MPI_SENDRECV`, are used for implementing parallel SMACOF, so that it is worth to investigate hybrid model SMACOF. For this dissertation, the author will analyze which parallel model is better-fitted to SMACOF algorithm.

6.3 Parallelization of MI-MDS

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 in Section 4 to embed the remaining points ($M = N - n$). Though

Table 2: 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	4	4
# Cores per node	16	24
Total Cores	128	768
L2 Cache per core	512 KB	2 MB
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

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.

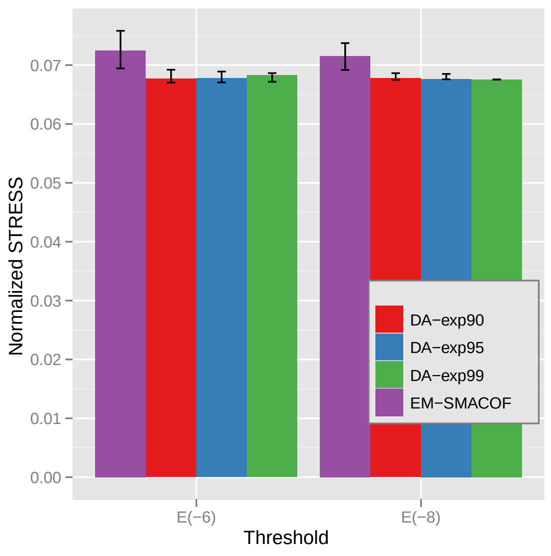
7 Performance Analysis

In this section, the author analyzes the experimental results of those proposed methods, such as parallelization, DA-SMACOF, and MI-MDS. All the applications are implemented in C# language and tested at multicore cluster systems in Table 2.

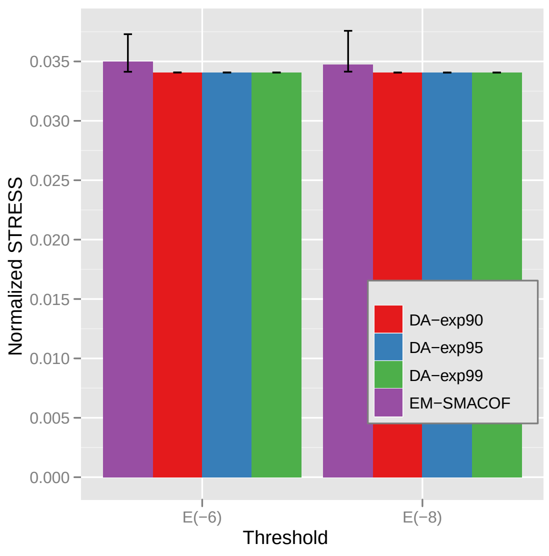
7.1 SMACOF vs. DA-SMACOF

In this section, the quality difference between EM-like SMACOF algorithm and the DA-SMACOF proposed in Section 3 is analyzed with respect to the well-known objective function STRESS (1). Biological sequence data, such as ALU sequence data and meta genomics sequence data, is used for the experiments. Note that it is difficult to find a vector representation of those biological sequence data, but the pairwise dissimilarity information between two different sequence is available.

The comparison between DA-SMACOF and EM-SMACOF with respect to the average mapping quality of 30 runs of ALU sequences with random initialization is illustrated in Fig. 3. There is a clear difference between EM-SMACOF and DA-SMACOF for both 2D and 3D mapping results which describes that every DA-SMACOF tests surpasses EM-SMACOF.



(a) ALU sequence 2D



(b) ALU sequence 3D

Figure 3: The normalized STRESS comparison between EM-SMACOF and DA-SMACOF with ALU sequence data with 3000 sequences for mapping in 2D and 3D space. Bar graph illustrates the average of 30 runs with random initialization and the corresponding error bar represents the minimum and maximum of the normalized STRESS value of EM-SMACOF and DA-SMACOF with different cooling parameters ($\alpha = 0.9$, 0.95, and 0.99). Note that the x-axis of both plots is the threshold value for the stop condition of SMACOF algorithm.

EM-SMACOF shows variation in quality for all experimental results. You should also note that the minimum of EM-SMACOF in 2D mapping results is clearly larger than average of all DA-SMACOF experiments. Even in 3D mapping comparison, the minimum of EM-SMACOF is still larger than the average of DA-SMACOF, although it seems to be similar to the average of DA-SMACOF in Fig. 3b. The results illustrates that the DA-SMACOF shows better and more reliable than the normal SMACOF with ALU sequence data.

Fig. 4 is the comparison between the average of 10 random initial runs of DA-SMACOF (DA-exp95) and EM-SMACOF with metagenomics data set. The threshold value of the stop condition for SMACOF algorithm is 10^{-8}. As expected, EM-SMACOF shows a tendency to be trapped in local optima by depicting some variation and larger STRESS values, and even the minimum values are bigger than any results of DA-exp95. In contrast to EM-SMACOF, all of the DA-exp95 results are very similar to each other. In fact, for 3D mappings, all of the DA-exp95 mappings reach at 0.0368854.

7.2 Parallel Performance of MPI-SMACOF

For the performance analysis of MPI-SMACOF discussed in Section 6.1, the author has applied the parallel algorithm for visualization of high-dimensional data into low-dimension to

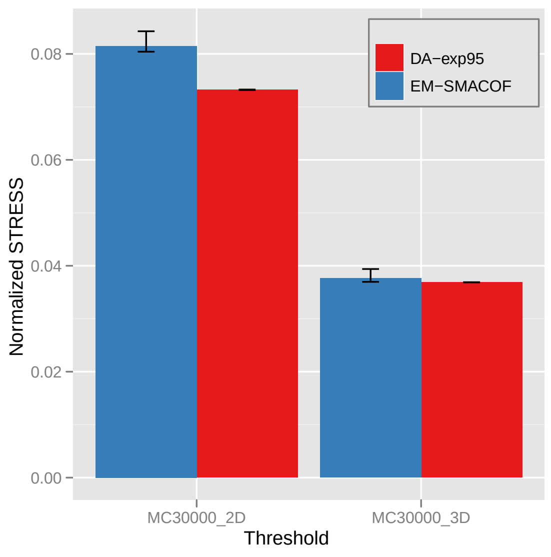


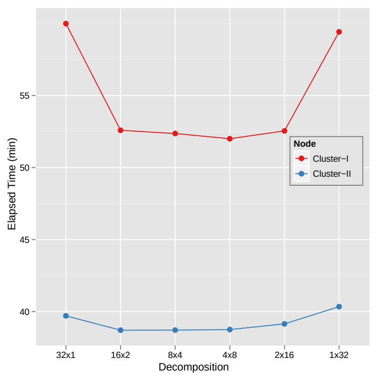
Figure 4: The normalized STRESS comparison between SMACOF and DA-SMACOF with Metagenomics sequence data with 30000 sequences for mapping in 2D and 3D space. Bar graph illustrates the average of 10 runs with random initialization and the corresponding error bar represents the minimum and maximum of the normalized STRESS value of EM-SMACOF and DA-SMACOF with $\alpha = 0.95$.

the dataset obtained from PubChem database[1], which is a NIH-funded repository for over 60 million chemical molecules and provides their chemical structure fingerprints and biological activities, for the purpose of chemical information mining and exploration. Among 60 Million PubChem dataset, the author has used randomly selected up to 100,000 chemical subsets and all of them have a 166-long binary value as a fingerprint, which corresponds to maximum input of 100,000 data points having 166 dimensions. With those data as inputs, we have performed our experiments on our two decent compute clusters as summarized in Table 2.

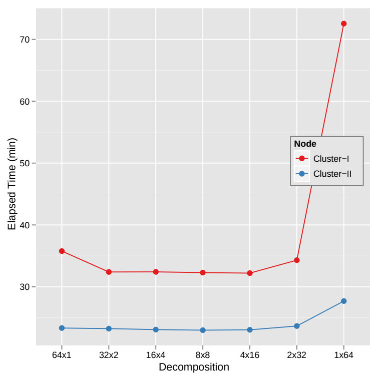
In the following, the performance results of the parallel SMACOF are shown with respect to 10,000, 20,000, 50,000 and 100,000 data points having 166 dimensions, represented as 10K, 20K, 50K, and 100K dataset respectively.

Fig. 5 shows the performance comparisons for 10K and 20K PubChem data with respect to how to decompose the given $N \times N$ matrices with 32, 64, and 128 cores in **Cluster-I** and **Cluster-II**. A significant characteristic of those plots in Fig. 5 is that skewed data decompositions, such as $p \times 1$ or $1 \times p$, which decompose by row-base or column-base, are always worse in performance than balanced data decompositions, such as $m \times n$ block decomposition which m and n are similar as much as possible. The reason of the above results is cache line effect that affects cache reusability, and generally balanced block decomposition shows better cache reusability so that it occurs less cache misses than the skewed decompositions [1, 21]. As in Fig. 5, Difference of data decomposition almost doubled the elapsed time of 1×128 decomposi-

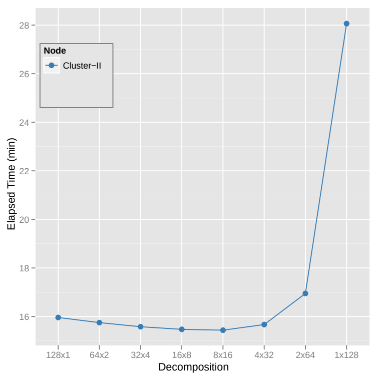
[1]`PubChem,http://pubchem.ncbi.nlm.nih.gov/`



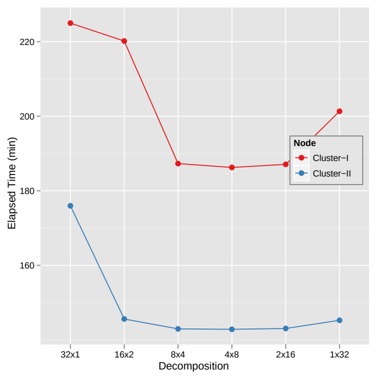
(a) 10K with 32 cores



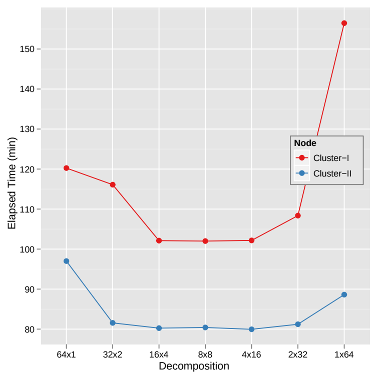
(b) 10K with 64 cores



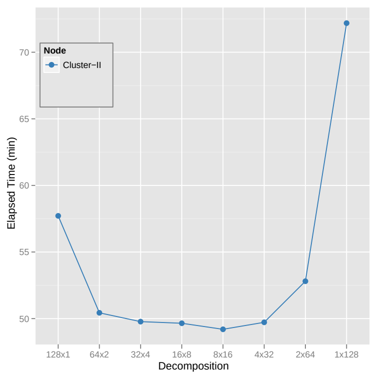
(c) 10K with 128 cores



(d) 20K with 32 cores



(e) 20K with 64 cores



(f) 20K with 128 cores

Figure 5: Performance of Parallel SMACOF for 10K and 20K PubChem data with 32,64, and 128 cores in **Cluster-I** and **Cluster-II** w.r.t. data decomposition of $N \times N$ matrices.

tion compared to 8×16 decomposition with 10K PubChem data. From the above investigation, it is derived that the balanced data decomposition is generally good choice. Furthermore, **Cluster-II** performs better than **Cluster-I** in Fig. 5, although the clock speed of cores is similar to each other. There are two different factors between **Cluster-I** and **Cluster-II** in Table 2 which we believe that those factors result in **Cluster-II** outperforms than **Cluster-I**, i.e. L2 cache size and Networks, and the L2 cache size per core is 4 times bigger in **Cluster-II** than **Cluster-I**. Since SMACOF with large data is memory-bound application, it is natural that the bigger cache size results in the faster running time.

In addition to data decomposition experiments, the parallel performance of pure MPI SMACOF is measured in terms of the number of processes p. The author investigates the scalability of parallel SMACOF by running with different number of processes, e.g. $p = 64$, 128, 256, and 384. On the basis of the above data decomposition experimental result, the balanced decomposition has been applied to this process scaling experiments. As p increases, the elapsed time should be decreased, but linear performance improvement could not be achieved due to the parallel overhead. In Fig. 6, both 50k and 100k data sets show the performance gain as p

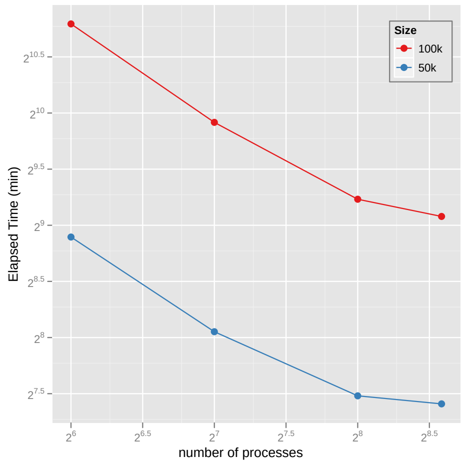


Figure 6: Performance of parallel SMACOF for 50K and 100K PubChem data in **Cluster-II** w.r.t. the number of processes. Based on the data decomposition experiment, we choose balanced decomposition as much as possible, i.e. 8×8 for 64 processes. Note that both x and y axes are log-scaled.

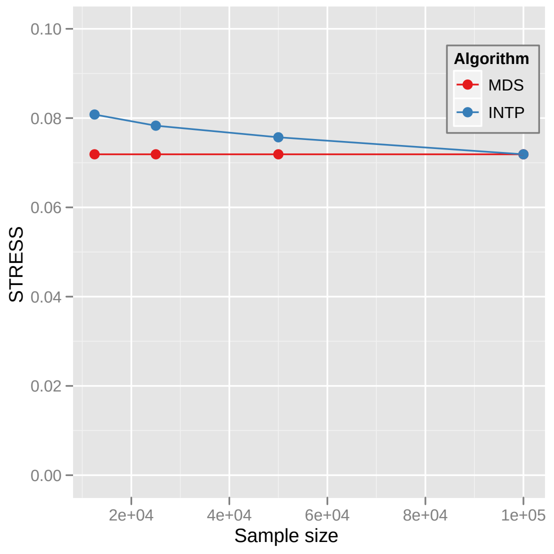
increases. However, performance enhancement ratio is reduced, because the ratio of message passing overhead over the assigned computation per each node increases due to more messaging and less computing per node as p increases. Note that we used 16 computing nodes in **Cluster-II** (total number of cores in 16 computing nodes is 384 cores) to perform the scaling experiment with large data set, i.e. 50k and 100k PubChem data, since SMACOF algorithm requires 480 GB memory for dealing with 100,000 data points, as we disscussed in Section 6.1, and **Cluster-II** is only feasible to perform this experiment with more than 10 nodes.

7.3 Interpolation Performance

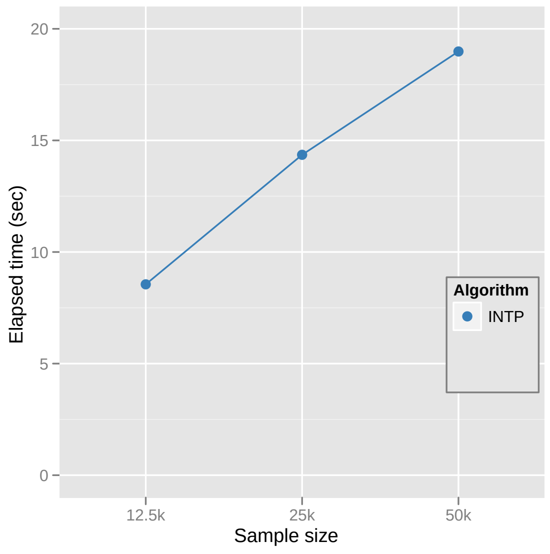
To measure the quality and parallel performance of the proposed MI-MDS discussed in Section 4, we have used 166-dimensional chemical dataset obtained from PubChem project database as in Section 7.2. In this section, the author has used randomly selected up to 2 million *out-of-sample* chemical subsets for interpolation testing.

In the following, The author will show i) the quality of the interpolation result of performing MI-MDS, and ii) performance measurement of the parallelized MI-MDS on the clustering systems as listed in Table 2.

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 algorithms show lower bias but higher variance with respect to smaller number of neighbors. 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, 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,



(a) normalized STRESS



(b) Elapsed Time

Figure 7: (a) Quality comparison between Interpolated result upto 100k based on the sample data and 100k MDS result. (b) Elapsed time of parallel MI-MDS upto 100k data w.r.t. the sample size using 16 nodes of the **Cluster-II** in Table 2. Note that the computational time complexity of MI-MDS is $\mathcal{O}(Mn)$ where n is the sample size and $M = N - n$.

the authors use 2-NN for the MI algorithm.

Fig. 7-(a) shows the comparison of quality between interpolated results upto 100K data with different sample size data 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-MDS algorithm produces good enough mapping in target dimension using very smaller amount of time than when we run MDS with full 100k data. Fig. 7-(b) shows the MI-MDS running time with respect to the sample data using 16 nodes of the **Cluster-II** in Table 2. The plot demonstrates that the running time is fitted to $\mathcal{O}(Mn)$. The running time of SMACOF with the sampled data (i.e. 12.5k, 25k, and 50k) and MI-MDS upto 100k data is substantially faster than running SMACOF with full 100k data. Note that the full MDS running time with 100k using 16 nodes of the **Cluster-II** in Table 2 is around 27006 sec.

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, 2M + 100k, and so on. The measured STRESS value is shown in Fig. 8. There are some quality lost between

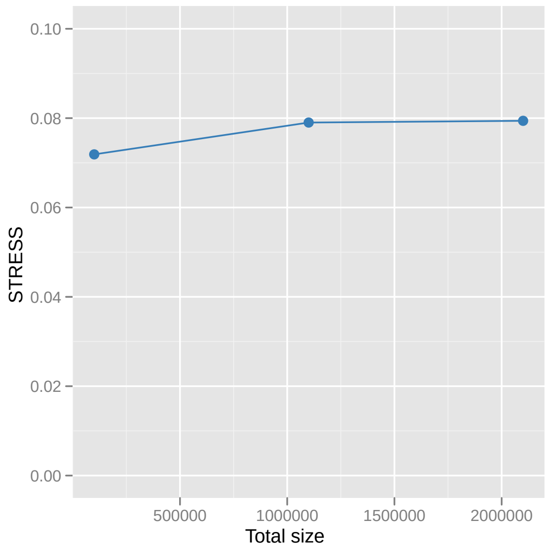


Figure 8: The normalized STRESS value of Interpolation larger data, such as 1M and 2M data points, with 100k sample data. The STRESS value of pre-mapped MDS result of 100k data is 0.0719.

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 2.

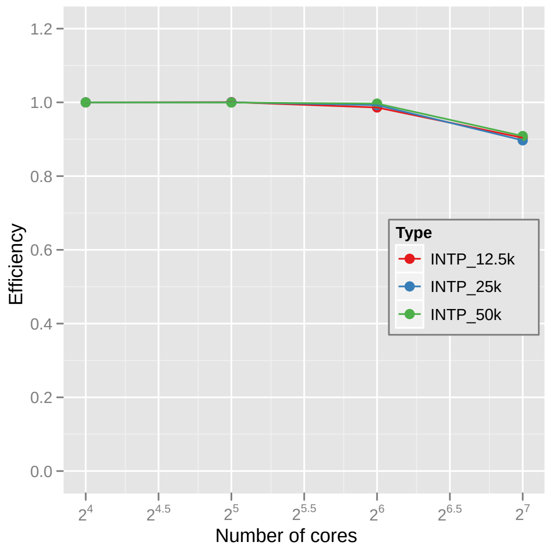
Here, the author would like to investigate the parallel performance of the proposed parallel MI-MDS implementation in terms of efficiency with respect to the running results within **Cluster-I** and **Cluster-II** in Table 2.

Both plots in Fig. 9 illustrate the efficiency 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:

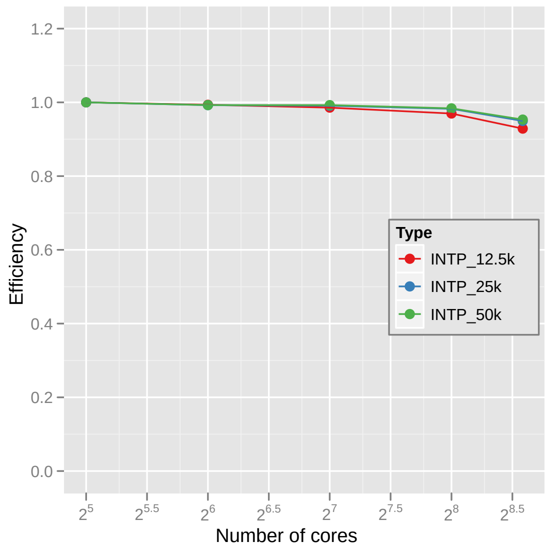
$$f = \frac{pT(p) - T(1)}{T(1)} \tag{38}$$

$$\varepsilon = \frac{1}{1+f} \tag{39}$$

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. (38) can be replaced with following:



(a) Cluster-I



(b) Cluster-II

Figure 9: Efficiency of runtime in parallel MI-MDS application with respect to different sample data size using **Cluster-I** and **Cluster-II** in Table 2. Total data size is 100K.

$$f = \frac{\alpha T(p_1) - T(p_2)}{T(p_2)} \tag{40}$$

where $\alpha = p_1/p_2$ and p_2 is the smallest number of used cores for the experiment, so $alpha \geq 1$. Here, Eq. (40) is used for calculating f in (39).

In Fig. 9-(a), 16 to 128 cores are used to measure parallel performance with 8 processes, and 32 to 384 cores are used to evaluate the performance of the proposed parallel MI-MDS with 16 processes in Fig. 9-(b). Processes communicate via MPI primitives and each process is also parallelized in thread level. Both Fig. 9-(a) and Fig. 9-(b) show very good efficiency with appropriate degree of parallelism.

8 Contributions

The main goal of this dissertation is to find low dimensional mapping of the given large high-dimensional data as good as possible and as many as possible using multicore cluster systems based on *pairwise dissimilarity* information. For this ultimate purpose, the author has proposed several methods to improve a well-known MDS algorithm, called SMACOF [7,8], with respect to both runtime and quality of mappings. Those efforts result in the following contributions. First, the author has applied DA approach [22, 23] to SMACOF algorithm so that it helps to prevent trapping local optima, and the experimental results in Section 7.1 verify that DA-SMACOF outperforms SMACOF in quality and shows consistent result. Second, both SMACOF and

DA-SMACOF algorithms are parallelized via pure MPI and hybrid parallel model so that it is possible not only to run faster but also to deal with larger data.

Though parallelization of SMACOF and DA-SMACOF enable us to run those algorithms with large data set using distributed memory resources, it is still difficult to run them with a huge amount of data, such as millions since those algorithms require $\mathcal{O}(N^2)$ memory. The interpolation algorithm called MI-MDS is proposed as an amendment of the above obstacle, in that MI-MDS is able to deal with millions of data points based on pre-mapped result of the subset of the given data (*in-sampled* data) without using $\mathcal{O}(N^2)$ computation and memory requirement. In Section 7.3, it is shown that the MI-MDS produces configuration of very large data with similar quality of running normal SMACOF. Last but not least, the proposed dissertation will investigate and analyze how weight values affect to the MDS mapping.

References

[1] Seung-Hee Bae. Parallel multidimensional scaling performance on multicore systems. In *Proceedings of the Advances in High-Performance E-Science Middleware and Applications workshop (AHEMA) of Fourth IEEE International Conference on eScience*, pages 695–702, Indianapolis, Indiana, Dec. 2008. IEEE Computer Society.

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

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

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

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

[7] Jan de Leeuw. Applications of convex analysis to multidimensional scaling. *Recent Developments in Statistics*, pages 133–146, 1977.

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

[9] 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.

[10] 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.

[11] David E. Goldberg. *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley, 1989.

[12] Thomas Hofmann and Joachim M. Buhmann. Pairwise data clustering by deterministic annealing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:1–14, 1997.

[13] John H. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, Ann Arbor, MI, 1975.

[14] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[15] Hansjörg Klock and Joachim M. Buhmann. Data visualization by multidimensional scaling: a deterministic annealing approach. *Pattern Recognition*, 33(4):651–669, 2000.

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

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

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

[19] Eliakim H. Moore. On the reciprocal of the general algebraic matrix. *Bulletin of American Mathematical Society*, 26:394–395, 1920.

[20] Roger Penrose. A generalized inverse for matrices. *Proceedings of the Cambridge Philosophical Society*, 51:406–413, 1955.

[21] Xiaohong Qiu, Geoffrey C. Fox, Huapeng Yuan, Seung-Hee Bae, George Chrysanthakopoulos, and Henrik Frystyk Nielsen. Data mining on multicore clusters. In *Proceedings of 7th International Conference on Grid and Cooperative Computing GCC2008*, pages 41–49, Shenzhen, China, Oct. 2008. IEEE Computer Society.

[22] Kenneth Rose. Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. *Proceedings of IEEE*, 86(11):2210–2239, 1998.

[23] Kenneth Rose, Eitan Gurewitz, and Geoffrey C. Fox. A deterministic annealing approach to clustering. *Pattern Recognition Letters*, 11(9):589–594, 1990.

[24] John W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, 18(5):401–409, 1969.

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

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

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

[28] Zizhuo Wang, Song Zheng, Yinyu Ye, and Stephen Boyd. Further relaxations of the semidefinite programming approach to sensor network localization. *SIAM Journal on Optimization*, 19(2):655–673, 2008.

[29] 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.