SCALABLE HIGH PERFORMANCE MULTIDIMENSIONAL SCALING

Seung-Hee Bae

Submitted to the faculty of the Graduate School

in partial fulfillment of the requirements

for the degree

Doctor of Philosophy

in the School of Informatics and Computing

Indiana University

February 2012

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements of the degree of Doctor of Philosophy.

Doctoral Committee

Geoffrey C. Fox
(Principal Advisor)

Randall Bramley

David B. Leake

January 17, 2012

David J. Wild

I dedicate this dissertation to my wife (Hee-Jung Kim) and my children (Seewon and Jian).

Acknowledgements

First of all, I am sincerely grateful to my advisor, Dr. Geoffrey C. Fox, for his insightful guidance and cheerful encouragement to this dissertation as well as my research projects. On the basis of his guidance and encouragement, it could be possible to complete Ph.D. degree. While I have been working with him, I could learn how to research as a scientist.

I would like to thank my research committee members: Dr. Randall Bramley, Dr. David Leake, and Dr. David Wild for their help, guidance, and invaluable comments to this dissertation. I would like to thank Dr. Sun Kim, who was my former advisor and had been my research committee member before he left Indiana University, for his valuable advices and encouragement.

It has been a pleasant time to work with those friendly and brilliant colleagues at Pervasive Technology Institute (PTI) for five years. I am thankful to Dr. Judy Qiu for her support and discussions for my research. I am also thankful to SALSA group members: Dr. Jaliya Ekanake, Thilina Gunarathne, Saliya Ekanayake, Ruan Yang, Hui Li, Tak-Lon Wu, Bingjing Zhang, Yuduo Zhou, Jerome Mitchell, Adam Hughes, and Scott Beason, for being fantastic lab mates. I am particularly thankful to Dr. Jong Youl Choi for countless valuable discussions on various research topics as well as non-technical topics during my time at PTI.

I would like to thank administrative staffs of the School of Informatics and Computing and PTI for their valuable helps to do my study at Indiana University (IU). I am grateful to Ms. Cathy McGregor Foster for

her helpful advice for Ph.D. study completion, and to Ms. Mary Nell Shiflet and Mr. Gary Miksik for their administrative support at PTI. Because of their supports, I could focus on my Ph.D. study while I has been at IU.

Last but not least, I am very much indebted to my family for their generous support and encouragement throughout my Ph.D. study at IU. My parents and my mother-in-law have been my big supporters during my Ph.D study, and their encouragement is always cheerful to finish this degree. I cannot fully explain my gratitude with any words to my lovely wife, Hee-Jung, for her endurance, encouragement, and support with love for my life as well as my study. Without her support and cheers, I would not make complete this long journey. Since I met her at Handong, she is the only perfect companion for my life. I would like to thank to my daughter (Seewon) and son (Jian) for being wonderful children to me. They are the source of my joy and happiness even though I was in a difficult time.

I am a debtor of love. Thank you, all!

Abstract

Today is so-called *data deluge era*. A huge amount of data is flooded in many domains of modern society based on the advancements of technologies and social networks. Dimension reduction is a useful tool for data visualization of such high-dimensional data and abstract data to make data analysis feasible for such large-scale high-dimensional or abstract scientific data. Among the known dimension reduction algorithms, multidimensional scaling (MDS) is investigated in this dissertation due to its theoretical robustness and high applicability. For the purpose of large-scale multidimensional scaling, we need to figure out two main challenges. One problem is that large-scale multidimensional scaling requires huge amounts of computation and memory resources, because it requires $\mathcal{O}(N^2)$ memory and computation. Another problem is that 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 this dissertation, to tackle two challenges mentioned above, we have applied three methodologies to multidimensional scaling: i) parallelization, ii) interpolation, and iii) deterministic annealing (DA) optimization. Parallelization is applied to provide required huge amounts of computation and memory resources by utilizing large-scale distributed-memory systems, such as multicore cluster systems. In addition, we have investigated an interpolation method which utilizes the known mappings of a subset of the given data, named *in-sample* data, to generate mappings of the remaining *out-of-sample* data. This approach dramatically reduces computational complexity and memory requirement. DA optimization method has been applied to

multidimensional scaling problem in order to avoid local optima. Experimental results illustrate the proposed methodologies are effective to scale up the mapping capacity of multidimensional scaling algorithm and to improve the mapping quality of multidimensional scaling via avoiding local optima.

Contents

Acknowledgements **v**

Abstract **vii**

1 Introduction **1**

1.1 Introduction 1

1.2 Multidimensional Scaling (MDS) 3

1.3 Motivation 5

1.4 The Research Problem 7

1.5 Contributions 8

1.6 Dissertation Organization 9

2 Backgrounds **11**

2.1 Classical Multidimensional Scaling 11

2.2 Scaling by a MAjorizing of a COmplicated Function (SMACOF) 14

2.3 Message Passing Interface (MPI) 16

2.4 Threading 18

2.5 Deterministic Annealing (DA) 19

3 High Performance Multidimensional Scaling **21**

3.1 Overview 21

3.2 High Performance Visualization 22

3.2.1 Parallel SMACOF 23

3.3 Performance Analysis of the Parallel SMACOF 30

3.3.1 Performance Analysis of the Block Decomposition 31

3.3.2 Performance Analysis of the Efficiency and Scalability 34

3.4 Summary 38

4 Interpolation Approach for Multidimensional Scaling **47**

4.1 Overview 47

4.2 Related Work 48

4.3 Majorizing Interpolation MDS 49

4.3.1 Parallel MI-MDS Algorithm 54

4.3.2 Parallel Pairwise Computation Method with Subset of Data 55

4.4 Analysis of Experimental Results 57

4.4.1 Exploration of optimal number of nearest neighbors 58

4.4.2 Comparison between MDS and MI-MDS 66
4.4.2.1 Fixed Full Data Case 66
4.4.2.2 Fixed Sample Data Size 68
4.4.3 Parallel Performance Analysis of MI-MDS 72
4.4.4 Large-Scale Data Visualization via MI-MDS 77
4.5 Summary 78

5 Deterministic Annealing SMACOF **80**

5.1 Overview 80
5.2 Related Work 81
5.2.1 Avoiding Local Optima in MDS 81
5.3 Deterministic Annealing SMACOF 83
5.3.1 Temperature Cooling Mechanism 87
5.4 Experimental Analysis 88
5.4.1 Iris Data 91
5.4.2 Chemical Compound Data 93
5.4.3 Cancer Data 95
5.4.4 Yeast Data 97
5.4.5 Running Time Comparison 98
5.5 Experiment Analysis of Large Data Sets 99

5.5.1 Metagenomics Data . . . 100
5.5.2 ALU Sequence Data . . . 103
5.5.3 16sRNA 50k Data . . . 104
5.5.4 16sRNA 100k Data . . . 106
5.5.5 Comparison of the STRESS progress . . . 109
5.5.6 Running Time Analysis of Large Data Sets . . . 111
5.6 Summary . . . 117

6 Conclusions and Future Works **119**

6.1 Summary of Work . . . 119
6.2 Conclusions . . . 120
6.2.1 High-Performance Distributed Parallel Multidimensional Scaling (MDS) . . . 121
6.2.2 Interpolation approach for MDS . . . 122
6.2.3 Improvement in Mapping Quality . . . 124
6.3 Future Works . . . 126
6.3.1 Hybrid Parallel MDS . . . 126
6.3.2 Hierarchical Interpolation Approach . . . 127
6.3.3 Future Works for DA-SMACOF . . . 127
6.4 Contributions . . . 128

Bibliography **131**

List of Tables

3.1 Main matrices used in SMACOF 23

3.2 Cluster systems used for the performance analysis 31

3.3 Runtime Analysis of Parallel Matrix Multiplication part of parallel SMACOF with 50k data set in Cluster-II 36

4.1 Compute cluster systems used for the performance analysis 59

4.2 Analysis of Maximum Mapping Distance between *k*-NNs with respect to the number of nearest neighbors (*k*). 62

4.3 Large-scale MI-MDS running time (seconds) with 100k sample data 70

List of Figures

1.1 An example of the data visualization of 30,000 biological sequences by an MDS algorithm, which is colored by a clustering algorithm. 5

3.1 An example of an $N \times N$ matrix decomposition of parallel SMACOF with 6 processes and 2×3 block decomposition. Dashed line represents where diagonal elements are. 25

3.2 Parallel matrix multiplication of $N \times N$ matrix and $N \times L$ matrix based on the decomposition of Figure 3.1 27

3.3 Calculation of $B(X^{[k-1]})$ matrix with regard to the decomposition of Figure 3.1. 29

3.4 Overall Runtime and partial runtime of parallel SMACOF for 6400 and 12800 PubChem data with 32 cores in Cluster-I and Cluster-II w.r.t. data decomposition of $N \times N$ matrices. 40

3.5 Overall Runtime and partial runtime of parallel SMACOF for 6400 and 12800 PubChem data with 64 cores in Cluster-I and Cluster-II w.r.t. data decomposition of $N \times N$ matrices. 41

3.6 Overall Runtime and partial runtime of parallel SMACOF for 6400 and 12800 PubChem data with 128 cores in Cluster-II w.r.t. data decomposition of $N \times N$ matrices. 42

3.7 Performance of parallel SMACOF for 50K and 100K PubChem data in Cluster-II w.r.t. the number of processes, i.e. 64, 128, 192, and 256 processes (cores). (a) shows runtime and efficiency is shown at (b). 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 for (a). 43

3.8 Efficiency of **tMatMult** and **tMM_Computing** in Table 3.3 with respect to the number of processes. 44

3.9 MPI Overhead of parallel matrix multiplication (**tMM_Overhead**) in Table 3.3 and the rough Estimation of the MPI overhead with respect to the number of processes. 45

3.10 Performance of parallel SMACOF for MC 30000 data in Cluster-I and Cluster-II w.r.t. the number of processes, i.e. 32, 64, 96, and 128 processes for Cluster-I and Cluster-II, and extended to 160, 192, 224, and 256 processes for Cluster-II. (a) shows runtime and efficiency is shown at (b). 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 for (a). 46

4.1 Message passing pattern and parallel symmetric pairwise computation for calculating STRESS value of whole mapping results. 56

4.2 Quality comparison between interpolated result of 100k with respect to the number of nearest neighbors (k) with 50k sample and 50k out-of-sample result. 60

4.3 The illustration of the constrained interpolation space when $k = 2$ or $k = 3$ by initialization at the center of the mappings of the nearest neighbors. 61

4.4 The mapping results of MI-MDS of 100k Pubchem data with 50k sample data and 50k out-of-sample data with respect to the number of nearest neighbors (k). The sample points are shown in red and the interpolated points are shown in blue. 63

4.5 Histogram of the original distance and the pre-mapping distance in the target dimension of 50k sampled data of 100k. The maximum original distance of the 50k sampled data is 10.198 and the maximum mapping distance of the 50k sampled data is 12.960. 64

4.6 Quality comparison between the interpolated result of 100k with respect to the different sample sizes (INTP) and the 100k MDS result (MDS) 67

4.7 Running time comparison between the Out-of-Sample approach which combines the full MDS running time with sample data and the MI-MDS running time with out-of-sample data when $N = 100k$, with respect to the different sample sizes and the full MDS result of the 100k data. 68

4.8 Elapsed time of parallel MI-MDS running time of 100k data with respect to the sample size using 16 nodes of the Cluster-II in Table 4.1. Note that the computational time complexity of MI-MDS is $\mathcal{O}(Mn)$ where n is the sample size and $M = N - n$. 69

4.9 The STRESS value change of the interpolation larger data, such as 1M, 2M, and 4M data points, with 100k sample data. The initial STRESS value of MDS result of 100k data is 0.0719. 70

4.10 Running time of the Out-of-Sample approach which combines the full MDS running time with sample data ($M = 100k$) and the MI-MDS running time with different out-of-sample data sizes, i.e. 1M, 2M, and 4M. 71

4.11 Parallel overhead modeled as due to MPI communication in terms of sample data size (m) using Cluster-I in Table 4.1 and message passing overhead model. 73

4.12 Parallel overhead modeled as due to MPI communication in terms of sample data size (m) using Cluster-II in Table 4.1 and message passing overhead model. 74

4.13 Efficiency of the interpolation part (INTP) and the STRESS evaluation part (STR) runtimes in the parallel MI-MDS application with respect to different sample data sizes using Cluster-I in Table 4.1. The total data size is 100K. 75

4.14 Efficiency of the interpolation part (INTP) and the STRESS evaluation part (STR) runtimes in the parallel MI-MDS application with respect to different sample data sizes using Cluster-II in Table 4.1. The total data size is 100K. 76

4.15 Interpolated MDS results of total 100k PubChem dataset trained by (a) 12.5k and (b) 50k sampled data. Sampled data are colored in red and interpolated points are in blue. 77

4.16 Interpolated MDS results. Based on 100k samples (a), additional 2M PubChem dataset is interpolated (b). Sampled data are colored in red and interpolated points are in blue. 78

5.1 The computational temperature movement with respect to two different cooling temperature mechanisms (**exponential** and **linear**). 89

5.2 The normalized STRESS comparison of the **iris** data mapping results in 2D space. The bar graph illustrates the average of 50 runs with random initialization, and the corresponding error bar represents the minimum and maximum of the normalized STRESS values of SMACOF, MDS-DistSmooth with different smoothing steps ($s = 100$ and $s = 200$) (**DS-s100** and **-s200** hereafter for short), and DA-SMACOF with different cooling parameters ($\alpha = 0.9$, 0.95, and 0.99) (**DA-exp90,-exp95**, and **-exp99** hereafter for short). The x-axis is the threshold value for the stopping condition of iterations (10^{-5} and 10^{-6}). 90

5.3 The 2D median output mappings of iris data with SMACOF (a), DS-s100 (b), and DA-exp95 (c), whose threshold value for the stopping condition is 10^{-5}. Final normalized STRESS values of (a), (b), and (c) are 0.00264628, 0.00208246, and 0.00114387, correspondingly. . . 92

5.4 The normalized STRESS comparison of the **chemical compound** data mapping results in 2D space. The bar graph illustrates the average of 50 runs with random initialization, and the corresponding error bar represents the minimum and maximum of the normalized STRESS values of SMACOF, DS-s100 and -s200, and DA-exp90, DA-exp95, and DA-exp99. The x-axis is the threshold value for the stopping condition of iterations (10^{-5} and 10^{-6}). 94

5.5 The normalized STRESS comparison of the **breast cancer** data mapping results in 2D space. The bar graph illustrates the average of 50 runs with random initialization, and the corresponding error bar represents the minimum and maximum of the normalized STRESS values of SMACOF, DS-s100 and -s200, and DA-exp90,DA-exp95, and DA-exp99. The x-axis is the threshold value for the stopping condition of iterations (10^{-5} and 10^{-6}). 96

5.6 The normalized STRESS comparison of the **yeast** data mapping results in 2D space. The bar graph illustrates the average of 50 runs with random initialization, and the corresponding error bar represents the minimum and maximum of the normalized STRESS values of SMACOF, DS-s100 and -s200, and DA-exp90,DA-exp95, and DA-exp99. The x-axis is the threshold value for the stopping condition of iterations (10^{-5} and 10^{-6}). 97

5.7 The average running time comparison between SMACOF, MDS-DistSmooth ($s = 100$), and DA-SMACOF (DA-exp95) for 2D mappings with tested data sets. The error bar represents the minimum and maximum running time. **EM-5/EM-6** represents SMACOF with $10^{-5}/10^{-6}$ threshold, and **DS-5/DS-6** and **DA-5/DA-6** represents the runtime results of MDS-DistSmooth and DA-SMACOF, correspondingly, in the same way. 98

5.8 The normalized STRESS comparison of the **metagenomics sequence** data mapping results in 2D and 3D space. The 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 values of SMACOF and DA-SMACOF with $\alpha = 0.95$. The x-axis is the threshold value for the iteration stopping condition of the SMACOF and DA-SMACOF algorithms (10^{-5} and 10^{-6}). 101

5.9 The normalized STRESS comparison of the **ALU sequence** data mapping results in 2D and 3D space. The 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 values of SMACOF and DA-SMACOF with $\alpha = 0.95$. The x-axis is the threshold value for the iteration stopping condition of the SMACOF and DA-SMACOF algorithms (10^{-5} and 10^{-6}). 102

5.10 The normalized STRESS comparison of the **16s RNA sequence** data with 50000 sequences for mapping in 2D and 3D space. The 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 values of SMACOF, DA-SMACOF with $\alpha = 0.95$ (**DA-exp95**), and DA-SMACOF with linear cooling with 100 steps (**DA-lin100**). The x-axis is the threshold value for the iteration stopping condition of the SMACOF and DA-SMACOF algorithms (10^{-5}, 10^{-6}, and *hybrid* approach). 104

5.11 The mapping progress of DA-SMACOF with **RNA50k** data set in 2D space with respect to computational temperature (T) and corresponding normalized STRESS value (σ). 107

5.12 The normalized STRESS comparison of the **16s RNA sequence** data with 100000 sequences for mapping in 2D and 3D space. The 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 values of SMACOF, DA-SMACOF with $\alpha = 0.95$ (**DA-exp95**), and DA-SMACOF with linear cooling with 100 steps (**DA-lin100**). The x-axis is the threshold value for the iteration stopping condition of the SMACOF and DA-SMACOF algorithms (10^{-5}, 10^{-6}, and *hybrid* approach). 108

5.13 The normalized STRESS progress comparison of **MC 30k** data in 2D and 3D space by SMACOF and DA-exp95. The x-axis is the cummulated iteration number of SMACOF and DA-SMACOF algorithms. 110

5.14 The normalized STRESS progress comparison of **RNA50k** data in 2D and 3D space by SMACOF and DA-exp95. The x-axis is the cummulated iteration number of SMACOF and DA-SMACOF algorithms. 112

5.15 The average running time comparison between SMACOF and DA-SMACOF (DA-exp95) for 2D and 3D mappings with MC30000 and ALU50058 data sets. The error bar represents the minimum and maximum running time. **EM-5/EM-6** represents SMACOF with $10^{-5}/10^{-6}$ threshold, and **DA-5/DA-6** represents the runtime results of DA-SMACOF, correspondingly, in the same way. 113

5.16 The average running time comparison between SMACOF, DA-SMACOF (DA-exp95), and DA-SMACOF with linear cooling (DA-lin100) for 2D and 3D mappings with 16s RNA data sets. The error bar represents the minimum and maximum running time. **EM-5/EM-6** represents SMACOF with $10^{-5}/10^{-6}$ threshold, and **DA-e5/DA-e5,6/DA-6** and **DA-l5/DA-l5,6/DA-l6** represents the runtime results of DA-exp95 and DA-lin100, correspondingly, with 10^{-5}/hybrid(10^{-5} and 10^{-6})/10^{-6}. 115

1

Introduction

1.1 Introduction

Because of the advancements of technology, a huge amount of data are produced in many domains of modern society, from digital personal information to scientific observation, and experimental data to medical records data. The current era could be referred to as the so-called *data deluge era*. In many scientific domains, the volumes of data are on the tera-scale and even the peta-scale. For the study of large-scale sky surveys in astronomy, about 20 terabytes of sky image data are collected by the Large Synoptic Survey Telescope [1] per night, and this phenomenon has led to about 60 petabytes of raw data over ten years of operations. In addition to large-scale sky survey data, biological sequence data has been produced in unimaginable volumes. Although the Human Genome Project [2], which was finished in 2003, was completed in 13 years and cost billions of dollars, now genome sequencing for an organism can be done much faster and more cheaply due to cost-effective high-throughput sequencing technologies. In fact, innovative sequencing technologies and the microarray technique have increased the volumes of biological data enormously.

[1] LSST: Large Synoptic Survey Telescope (http://www.lsst.org/lsst/)
[2] Human Genome Project (http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml)

In 2005, Intel proposed the Recognition, Mining, and Synthesis (RMS) [24] approach as a killer application for the next data explosion era. Machine learning and data mining algorithms were suggested as important algorithms for the data deluge era by [24]. Mining some meaningful information from these large volumes of raw data requires a huge amount of computing power which exceeds a single computing machine. To make matters worse, the asymptotic time complexities of most of the data mining algorithms are larger than a simple $\mathcal{O}(N)$, in that they require a significant amount of processing capability for analyses over large volumes of data. Thus, parallel and distributed computing is a critical feature of performing such data analyses. The efficiency and the scalability should be achieved in a parallel implementation of algorithms in order to maximize the effects of parallel and distributed computing.

One of the innovative inventions that has emerged in the computer hardware community during the last decade was the invention of multi-core architecture. The classical method of improving the computing power of computing processing units (CPUs), such as increasing clock speed, has been limited by physical obstacles; therefore, the CPU companies changed the focus of improving computing power from increasing the clock speed of a CPU to increasing the number of cores in a CPU chip. Since multicore architecture as invented, multicore architecture has become important in software development with effects on the client, the server and supercomputing systems [5, 23, 24, 66]. As mentioned in [66], the parallelism has become a critical issue for developing software for the purpose of effectively using multicore systems.

From the above statements, the necessary computation will be enormous for data mining algorithms in the future, and classical sequential programming schemes will not be suitable for multicore systems any more, so that implementing in scalable parallelism these algorithms will be one of the most important procedures for the coming many-core and data explosion era.

Among the many data mining areas which exist, such as clustering, classification, and association rule mining, and so on, dimension reduction algorithms are used to visualize high-dimensional data or abstract data into a low-dimensional target space. Dimension reduction algorithms can be used for data visualization

which can be applied to usage to fulfill the following purposes: (1) representing unknown data distribution structures in human-perceptible space with respect to the pairwise proximity or topology of the given data; (2) verifying a certain hypothesis or conclusion related to the given data by showing the distribution of the data; and (3) investigating relationship among the given data by spatial display.

Among the known dimension reduction algorithms, such as Principal Component Analysis (PCA), Multidimensional Scaling (MDS) [13, 45], Generative Topographic Mapping (GTM) [11, 12], and Self-Organizing Maps (SOM) [43], to name a few, multidimensional scaling (MDS) has been extensively studied and used in various real application areas, such as biology [48, 71], stock market analysis [33], computational chemistry [4], and breast cancer diagnosis [46]. This dissertation focuses on the MDS algorithm, and we investigate two ultimate goals: (1) how to achieve the scalability of the MDS algorithm to deal with large-scale data; and (2) how to improve the mapping quality of MDS solutions as well as the scalable MDS. This dissertation also describes some detailed performance analyses and experiments related to the proposed methodologies.

1.2 Multidimensional Scaling (MDS)

Multidimensional scaling (MDS) [13, 45, 68] is a general term that refers to techniques for constructing a map of generally high-dimensional data into a target dimension (typically a low dimension) with respect to the given pairwise proximity information. Mostly, MDS is used to visualize given high dimensional data or abstract data by generating a configuration of the given data which utilizes Euclidean low-dimensional space, i.e. two-dimension or three-dimension.

Generally, proximity information, which is represented as an $N \times N$ dissimilarity matrix ($\Delta = [\delta_{ij}]$), where N is the number of points (objects) and δ_{ij} is the dissimilarity between point i and j, is given for the MDS problem, and the dissimilarity matrix (Δ) should agree with the following constraints: (1) symmetricity ($\delta_{ij} = \delta_{ji}$), (2) nonnegativity ($\delta_{ij} \geq 0$), and (3) zero diagonal elements ($\delta_{ii} = 0$). The objective of the MDS

technique is to construct a configuration of a given high-dimensional data into low-dimensional Euclidean space, where each distance between a pair of points in the configuration is approximated to the corresponding dissimilarity value as much as possible. The output of MDS algorithms could be represented as an $N \times L$ configuration matrix X, whose rows represent each data point x_i ($i = 1, \ldots, N$) in L-dimensional space. It is quite straightforward to compute the Euclidean distance between x_i and x_j in the configuration matrix X, i.e. $d_{ij}(X) = \|x_i - x_j\|$, and we are able to evaluate how well the given points are configured in the L-dimensional space by using the suggested objective functions of MDS, called STRESS [44] or SSTRESS [67]. Definitions of STRESS (1.1) and SSTRESS (1.2) are following:

$$\sigma(X) = \sum_{i<j\leq N} w_{ij}(d_{ij}(X) - \delta_{ij})^2 \tag{1.1}$$

$$\sigma^2(X) = \sum_{i<j\leq N} w_{ij}[(d_{ij}(X))^2 - (\delta_{ij})^2]^2 \tag{1.2}$$

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

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

Figure 1.1 is an example of the data visualization of 30,000 biological sequence data, which is related to a metagenomics study, by an MDS algorithm. The colors of the points in Figure 1.1 represent the clusters of the data, which is generated by a pairwise clustering algorithm by deterministic annealing [36]. The data visualization in Figure 1.1 shows the value of the dimension reduction algorithms which produced lower dimensional mapping for the given data. We can see clearly the clusters without quantifying the quality of the clustering methods statistically.

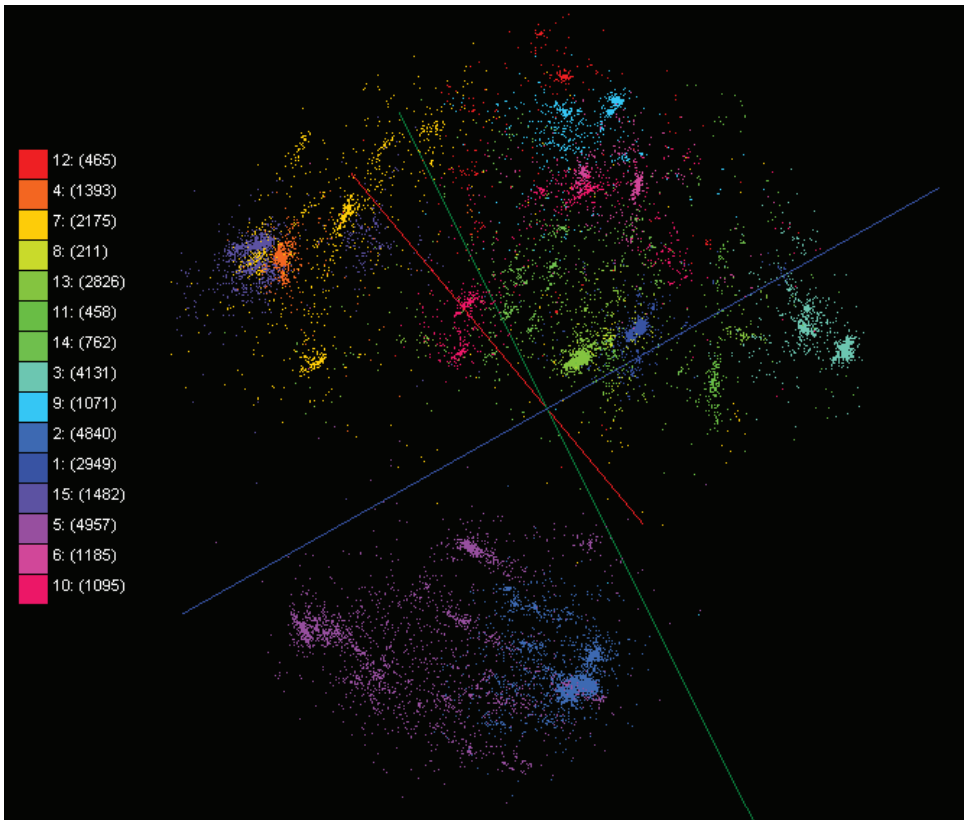


Figure 1.1: An example of the data visualization of 30,000 biological sequences by an MDS algorithm, which is colored by a clustering algorithm.

1.3 Motivation

The recent explosion of publicly available biological gene sequences, chemical compounds, and various scientific data offers an unprecedented opportunity for data mining. Among the various available data mining algorithms, dimension reduction is a useful tool for information visualization of high-dimensional data to make analysis feasible for large volume and high-dimensional scientific data. It facilitates the investigation of unknown structures of high dimensional data in three (or two) dimensional visualization.

In contrast to other algorithms, like PCA, GTM, and SOM, which generally construct a low dimensional configuration based on vector representations of the data, MDS aims at constructing a new mapping in a target

dimension on the basis of pairwise proximity (typically dissimilarity or distance) information; as a result, it does not require feature vector information of the underlying application data to acquire a lower dimensional mapping of the given data. Hence, MDS is an extremely useful approach for data visualization of a certain type of data, which would prove impossible or improper to represent by feature vectors but tat has pairwise dissimilarity, such as a biological sequence data. MDS, of course, is also applicable to data represented by feature vectors as well. MDS provides more broad applicability than other dimension reduction methods in terms of the given format of the data.

In the past, the data size given dealt with by machine learning and data mining algorithms was usually small enough to be executed on a single CPU or node without any consideration of parallel computing. On the other hand, in the modern data-deluged world, we can find many interesting data sets with large amounts of units which are impossible to run as sequential programming on a single node due, not only to the prohibitive computing requirements but also, because of the required memory size. Therefore, we have to figure out how to increase the computational capability of the MDS algorithm in order to manage large-scale high-dimensional data sets. In addition to the increase of data size, the invention and emergence of multi-core architectures has also required a change in the programming paradigm, so as to be able to utilize the maximal performance of the multi-core chips.

In general, there are two different approaches with which to improve the computing capability of MDS algorithm. One is to increase the available computing resources, i.e. CPUs and main memory size, and the other method is to reduce the computational time complexity.

In addition to the motivation of increasing computational capability because of the large scale of the data, another motivation of this thesis is based on the fact that the MDS problem is a non-linear optimization problem, which means it might have many local optima, so the avoidance of non-global local optima is an essential property needed in order to obtain a better quality of MDS outputs. Also, if we achieve the local optima avoidance feature, it may result in generating less sensitive parameters and more consistent MDS

output, than the output of the MDS algorithm which suffers from being trap in local optima. Thus, this dissertation also investigates how to avoid local optima effectively and efficiently.

1.4 The Research Problem

The explosion of data and the invention of multi-core architecture has brought forth interesting reseach issues in the data mining community as well as other computer science areas. We reviewed the interesting research motivations in relation to the MDS algorithm in Section 1.3. As we discussed in Section 1.3, two main goals of this dissertation are: (1) examining the scalability of the MDS algorithm due to the large-scale of the given data, i.e. millions of points; and (2) the local optima avoidance issue of MDS, which is a non-linear optimization problem. These motivations of this dissertation have lead us to the following research problems:

Parallelization

Applying distributed parallelism to the MDS algorithm is a natural process for achieving an increase in the computational capability by using distributed computing nodes. This makes it possible to acquire more computing resources and to utilize the full power of multi-core systems. Several issues should be covered to implement an algorithm in parallel, such as the load-balance, efficiency, and scalability issues.

Reduction of Complexity

Since the time complexity and memory requirements of the MDS algorithm is $\mathcal{O}(N^2)$ due to the use of an N by N pairwise dissimilarity matrix, where N is the number of points, and the distributed parallel implementation of MDS algorithm is still limited to the number of available computing resources, the development of an innovative MDS method, which reduces the time complexity and the memory

requirement, is required to reach the goal of scalability of the MDS algorithm. The parallel implementation of the proposed MDS approach will be highly encouraged because the size of the data could be enormous.

Optimization Method

Trapping in a local optima problem is a well-known problem of the non-linear optimization methods, including MDS problem. This is a very important issue for the non-linear optimization problems to avoid local optima. However, it is difficult to avoid local optima by using simple heuristics, which are based on the hill-climbing optimization method. Various non-determinstic optimization methods, such as the Genetic Algorithms (GA) [37] and Simulated Annealing (SA) [40], were proposed as solutions of local optima avoidance problems. They have been used for many non-linear optimization problems. As with other data mining algorithms, various optimization methods have been applied to the MDS problem for the purpose of avoiding local optima. It is true that the GA and SA algorithms are very successful for avoiding local optima, which is the goal of these algorithms. These algorithms, however, are also known to suffer from long running times due to their non-deterministic random walking approach. By taking the above statements into consideration, we would like to solve the local optima issue of the MDS problem by applying a deterministic optimization method, which is not based on a simple gradient descent approach.

1.5 Contributions

We can summarize the contributions of this dissertation as being the purpose of improving MDS algorithms in relation to computational capability and mapping quality such as the following:

- **[Parallelization]** Efficient parallel implementation via the Message Passing Interface (MPI) in order to scale up the computing capability of a well-known MDS algorithm (SMACOF) by utilizing distributed

systems, i.e. cluster systems.

- **[Reducing Complexity]** Development of an innovative algorithm, which reduces the computational complexity and memory requirement of MDS algorithms, and which produces acceptable mapping results; this step will be taken for the purpose of scaling the MDS algorithm's capacity up to millions of points, a step which is usually intractable for generating a mapping via normal MDS algorithms.
- **[Local Optima Avoidance]** Providing an MDS algorithm which could comprehend out the local optima avoidance problem in a deterministic way so that it generates better quality mapping in a reasonable amount of time.

1.6 Dissertation Organization

This dissertation is composed of several chapters and each chapter describes a unique part of this thesis as follows:

In Chapter 2, we describe some background methodologies related to this dissertation. First, we discussed the well-known MDS algorithm which is named Scaling by a MAjorizing of a COmplicated Function (SMACOF). This is implemented in distributed parallel fashion in Chapter 3. Also, we summarize the optimization method, named Deterministic Annealing (DA), which aims at avoiding local optima and which is used in this dissertation. The Message Passing Interface (MPI) standard is briefly mentioned in this chapter as well.

How to achieve distributed parallel implementation of the SMACOF algorithm is illustrated in detail in Chapter 3. Since the load balance has a critical impact on the efficiency of the parallel implementations, we discuss how to decompose and spread out the given data to each process; this approach is directly connected to the load balance issue. Furthermore, we described the message passing patterns of each components of the parallelized algorithm, here SMACOF, in Chapter 3. The detailed experimental analysis, which is based

on the testing results on two different cluster systems, follows the explanation of the parallel implementation in Chapter 3.

Chapter 4 describes the interpolation algorithm of the MDS problem which reduces computational complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(nM)$, where N is the full data size, n is the *sampled* data size, and $M = N - n$. Furthermore, we introduce how to parallelize the proposed interpolation algorithm of MDS due to the huge amounts of data points. This section is followed by a discussion of a quality comparison between the proposed interpolation algorithm and the full MDS algorithm, as well as parallel performance analysis of the parallelized implementation of this algorithm.

In contrast to Chapter 3 and Chapter 4, which focus on scaling up the computing capability of MDS algorithms, we propose the step of applying the deterministic annealing (DA) [62, 63] optimization method to the MDS problem; this approach will result in an avoidance of local optima in a deterministic way. This information is presented in Chapter 5. In Chapter 5, we also compare the DA applied MDS algorithm with other MDS algorithms, with respect to the mapping quality and running time.

Finally, in Chapter 6, we present the conclusions of this dissertation and future research interests related to the proposed algorithms discussed in this dissertation.

2

Backgrounds

2.1 Classical Multidimensional Scaling

The *classical scaling* (Classical MDS) was the first practical technique available for MDS, which was proposed by Torgerson [68, 69] and Gower [31]. The linear algebraic theorems mentioned by Eckart and Young [25] and Young and Householder [75] are the bases of the classical MDS. The main idea of the classical MDS is that, if the dissimilarities are represented by Euclidean distances, then we can find the configurations which represent the dissimilarities by using some matrix operations. Here, we briefly introduce the classical MDS algorithm, and the algorithm is explained in detail in Chapter 12 of [13].

The matrix of squared Euclidean distances of the given coordinates ($D^{(2)}(X)$ or simply $D^{(2)}$) can be expressed by a simple matrix equation with respect to the coordinate matrix (X), as shown in (2.1) and (2.2):

$$D^{(2)} = c1^t + 1c^t - 2XX^t \tag{2.1}$$

$$= c1^t + 1c^t - 2B, \tag{2.2}$$

where c is the diagonal elements of XX^t, 1 is the one vector whose elements are all ONE, 1^t, c^t, and X^t are transpose of 1, c, and X, correspondingly, and $B = XX^t$. (2.1) illustrates the relation between the squared distances and the scalar products of the coordinate matrix (X).

The centering matrix (J) can be defined as $J = I - n^{-1}11^t$, where I is the identity matrix, which translates a matrix to a column centered matrix by multiplying them. By multiplying the left and the right sides by the centering matrix J, a process called the *double centering* operation, we can introduce the following equations:

$$\begin{aligned} JD^{(2)}J &= J(c1^t + 1c^t - 2XX^t)J && (2.3)\\ &= Jc1^tJ + J1c^tJ - J(2B)J && (2.4)\\ &= Jc0^t + 0c^tJ - 2JBJ && (2.5)\\ &= -2JBJ && (2.6)\\ &= -2B. && (2.7) \end{aligned}$$

Since the centering of a vector of ones turns out to be a vector of zeros ($1^tJ = J1 = 0$), the first two terms are elliminated. Without a loss of generality, we can assume that the coordinate matrix (X) is a column centered matrix. Then, the result of the double centering operation on the B matrix is equal to B itself, since X is a column centered matrix. Therefore, we can define the relation between B and $D^{(2)}$ as in (2.8).

$$B = -\frac{1}{2}JD^{(2)}J. \quad (2.8)$$

If we apply *eigendecomposition* on B, we can have $B = Q\Lambda Q^t$. Λ could be represented as $\Lambda^{1/2}\Lambda^{1/2}$, and if we apply this to the eigendecomposition results of the B matrix:

Algorithm 2.1 Classical MDS algorithm

1: Calculate the matrix of squared dissimilarity $\Delta^{(2)}$.
2: Compute B_Δ by applying double centering: $B_\Delta = -\frac{1}{2} J \Delta^{(2)} J$.
3: Compute the eigendecomposition of $B_\Delta = Q \Lambda Q^t$.
4: /* Q_+ is the first L eigenvectors of Q */
5: /* Λ_+ is the first L eigenvalues of Λ which is greater than ZERO. */
6: Calculate L-dimensional coordinate matrix X by $X = Q_+ \Lambda_+^{1/2}$.
7: **return** X.

$$
\begin{aligned}
B &= Q \Lambda Q^t && (2.9)\\
&= Q \Lambda^{1/2} \Lambda^{1/2} Q^t && (2.10)\\
&= Q \Lambda^{1/2} (\Lambda^{1/2})^t Q^t && (2.11)\\
&= (Q \Lambda^{1/2})(Q \Lambda^{1/2})^t = X X^t. && (2.12)
\end{aligned}
$$

By (2.12), we can find the coordinate matrix from the given squared distance matrix. The classical MDS is very close to the above method, and the only difference is that it uses the squared dissimilarity matrix ($\Delta^{(2)}$) instead of the matrix of squared distances ($D^{(2)}$).

Algorithm 2.1 describes the classical MDS procedure, and its computational time complexity is $\mathcal{O}(N^3)$ due to the eigendecomposition. Note that, if Δ is not a Euclidean distance matrix, some eigenvalues could be negative. The classical MDS algorithm ignores those negative eigenvalues as errors. Since the classical MDS is an analytical solution, it does not require iterations to get a solution as shown in Algorithm 2.1. Another merit of the classical MDS is that it provides nested solutions, in that the two dimensions of a 2D solution are the same as the first two dimensions of a 3D result. The objective function of the classical MDS is called *STRAIN*, and it is defined as follows:

Algorithm 2.2 SMACOF algorithm

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

$$S(X) = ||XX^t - B_{\Delta}||^2. \tag{2.13}$$

2.2 Scaling by a MAjorizing of a COmplicated Function (SMACOF)

There are a lot of different algorithms which could be used to solve the MDS problem, and Scaling by MAjorizing a COmplicated Function (SMACOF) [20, 21] is one of them. SMACOF is an iterative majorization algorithm used to solve the MDS problem with the STRESS criterion. The iterative majorization procedure of the SMACOF could be thought of as an Expectation-Maximization (EM) [22] approach. Although SMACOF has a tendency to find local minima due to its hill-climbing attribute, it is still a powerful method since the algorithm, theoretically, guarantees a decrease in the STRESS (σ) criterion monotonically. Instead of a mathematically detail explanation of the SMACOF algorithm, we illustrate the SMACOF procedure in this section. For the mathematical details of the SMACOF algorithm, please refer to [13].

Algorithm 2.2 illustrates the SMACOF algorithm for the MDS solution. The main procedure of the SMACOF are its iterative matrix multiplications, called the *Guttman transform*, as shown in Line 8 in Algorithm 2.2, where $V^{\dagger}$ is the Moore-Penrose inverse [52, 54] (or pseudo-inverse) of matrix V. The $N \times N$ matrices V and $B(Z)$ are defined as follows:

$$V = [v_{ij}] \tag{2.14}$$

$$v_{ij} = \begin{cases} -w_{ij} & \text{if } i \neq j \\ \sum_{i \neq j} w_{ij} & \text{if } i = j \end{cases} \tag{2.15}$$

$$B(Z) = [b_{ij}] \tag{2.16}$$

$$b_{ij} = \begin{cases} -w_{ij}\delta_{ij}/d_{ij}(Z) & \text{if } i \neq j \\ 0 & \text{if } d_{ij}(Z) = 0, i \neq j \\ -\sum_{i \neq j} b_{ij} & \text{if } i = j \end{cases} \tag{2.17}$$

If the weights are equal to one ($w_{ij} = 1$) for all pairwise dissimilarities, then V and $V^{\dagger}$ are simplified as follows:

$$V = N\left(I - \frac{ee^t}{N}\right) \tag{2.18}$$

$$V^{\dagger} = \frac{1}{N}\left(I - \frac{ee^t}{N}\right) \tag{2.19}$$

where $e = (1, \ldots, 1)^t$ is one vector whose length is N. In this thesis, we generate mappings based on the equal weights weighting scheme and we use (2.19) for $V^{\dagger}$.

As in Algorithm 2.2, the computational complexity of the SMACOF algorithm is $\mathcal{O}(N^2)$, since the Guttman transform performs a multiplication of an $N \times N$ matrix and an $N \times L$ matrix twice, typically $N \gg L$ ($L = 2$ or 3), and the computation of the STRESS value, $B(X^{[k]})$, and $D(X^{[k]})$ 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 3.1. Due to the trends of digitization, data sizes have increased enormously, so it is critical that we are able to investigate large data sets. However, it is impossible to run SMACOF for a large data set under a typical

single node computer due to the memory requirement increases in $\mathcal{O}(N^2)$. In order to remedy the shortage of memory in a single node, we illustrate how to parallelize the SMACOF algorithm via message passing interface (MPI) for utilizing distributed-memory cluster systems in Chapter 3.

2.3 Message Passing Interface (MPI)

The Message Passing Interface (MPI) [27,73] standard is a language-independent message passing protocol, and is the one of the most widely used parallel programming methods in the history of parallel computing. MPI is a library specification for a message passing system, which aims at utilizing distributed computing resources, i.e. computer cluster systems, for the purpose of increasing the computing power to deal with the large scale problem caused by the communication between processes via messages.

MPI specification is composed of the definitions of a set of routines used to illustrate various parallel programming models effectively, such as point-to-point communication, collective communication, the topologies of communicators, derived data types, and parallel-IO. Since MPI uses language-independent specifications for calls and language bindings, MPI runtimes are available for many programming languages, for instance, Fortran, C, C++, Java, C#, Python, and so on.

MPI communication procedures can be categorized by two different mechanisms: (i) a point-to-point communication procedure in which a communication is operated between two processes; and (ii) a collective communication procedure, in which all processes within a communication group should invoke the collective communication procedure.

A point-to-point communication can occur with a pair of *send* and *receive* operations. The MPI send and receive operations are divided into: (i) a blocking operation and (ii) a nonblocking operation. In [27], blocking and nonblocking operations are defined as follows:

blocking If a return from the procedure indicates the user is allowed to re-use resources specified in the call.

nonblocking If the procedure may return before the operation completes, and before the user is allowed to re-use resources (such as buffers) specified in the call.

In addition to a *standard* mode, MPI also proposes three more communication modes: the (i) *buffered*, (ii) *synchronous*, and (iii) *ready* modes. In the **standard** communication mode, MPI decides whether the outgoing message will be buffered or not, not by users, so that a stardard mode send operation can be started regardless of posting of the matching receive; this command may be complete before a maching receive is posted, or it will not complete until a maching receive has been posted and the data has been moved to the receiver. A **buffered** mode send operation is defined as a **local** procedure so that it depends only on the local process and can complete without regard to other processes. In fact, MPI must buffer the outgoing message of a buffered mode send operation so it can complete in the local region. In the **synchronous** communication mode, a send operation can be started regardless of posting of the matching receive, similar to the standard and buffered modes. However, it will not complete successfully until a matching receive operation is posted and the receive operation has started to receive the message of the synchronous send. A send in the **ready** mode may be started *only if* the matching receive operation is already posted, unlike a send operation in other communication modes. Both blocking and nonblocking send operations can use these four different communication modes explained above.

Several collective communication procedures are defined in the MPI standard [27]: barrier synchronization, broadcast, scatter, gather, allgather, scatter/gather (which is complete exchange or all-to-all), and reduction/allreduction operations i.e. sum, min, max, or user defined functions. A collective operation requires that all processes in the communication group call the communication routine.

MPI is highly used under distributed cluster systems, which connected via a high-speed network; it is used for utilizing large-scale computing and memory resources, provided by these cluster systems, to solve

large-scale data-intensive or computing-intensive applications. MPI also supports various communication topologies, such as 2D or 3D grids and general graphs topologies, as well as dynamic communiation groups. In addition, new types of functionality, such as dynamic processes, one-sided communication, parallel I/O, and so on, are added to MPI standard 2 [26]. MPI provides flexible fine-grained parallel programming environment based on various features of MPI.

2.4 Threading

Emerging multicore processors places a spotlight parallel computing, including threading, since it is able to supply many computing units in a single node and even in a single CPU. Threading is used to investigate parallelism within shared memory systems, such as graphics processors, multicore systems, and Symmetric Multiprocessor (SMP) Systems.

Threading supports fine grained task parallelism, which could be effective for various applications without message passing overhead. For the correct use of threading, however, we have to consider the following: (i) dealing with critical sections, which should be mutually exclusive between threads; and (ii) cache false sharing overhead and cache line effects. There are a lot of threading libraries, which suppport parallelism via threads, such as POSIX Threads [15], OpenMP [3], the Task Parallel Library (TPL) [47], Intel Threading Building Blocks (TBB) [2, 60], and the Boost library [1, 39], to name a few. The Concurrency and Coordination Runtime (CCR) [18, 61] library supports a more complicated threading parallelism through message passing via Port, which can be thought of as a way to queue messages. In addition to these libraries, most programming languages also support threading in various forms.

Since multicore technology has been invented, multicore CPUs have become universal and typical, in that most cluster systems are multicore cluster systems. Multicore cluster systems are distributed memory systems which are composed of multiple shared memory system nodes by network connection. Thus, Hybrid parallel

programming paradigms, which combine distributed memory parallelism via MPI for inter-node communications and shared memory parallelism via threading libraries, i.e. OpenMP and TPL, within each node, have been investigated [56, 59]. In this dissertation, I have used the hybrid parallel paradigm in Chapter 4 as well.

2.5 Deterministic Annealing (DA)

Since the simulated annealing (SA) was introduced by Kirkpatrick et al. [40], people widely accepted SA and other stochastic maximum entropy approaches to solve optimization problems for the purpose of finding global optimum instead of hill-climbing deterministic approaches. SA is a Metropolis algorithm [51], which accepts not only the better proposed solution but even the worse proposed solution than the previous solution, based on a certain probability which is related to the *computational temperature* (T). Also, it is known that the Metropolis algorithm converges to an equilibrium probability distribution known as the *Gibbs probability distribution*. If we denote $\mathscr{H}(X)$ as the energy (or cost) function and $\mathscr{F}$ as a *free energy*, then Gibbs distribution density as follows:

$$P^G(X) = \exp\left(-\frac{1}{T}(\mathscr{H}(X) - \mathscr{F})\right), \tag{2.20}$$

$$\mathscr{F} = -T\log\int \exp\left(-\frac{1}{T}\mathscr{H}(X)\right)\mathrm{d}X. \tag{2.21}$$

and the *free energy ($\mathscr{F}$)*, which is a suggested objective function of SA, is minimized by the Gibbs probability density P^G. Also, free energy $\mathscr{F}$ can be written as follows:

$$\mathscr{F}_P = <\mathscr{H}>_P - T\mathscr{S}(P) \tag{2.22}$$

$$\equiv \int P(X)\mathscr{H}(X)\mathrm{d}X + T\int P(X)\log P(X)\mathrm{d}X \tag{2.23}$$

where $< H >_P$ represents the *expected energy* and $\mathscr{S}(P)$ denotes *entropy* of the system with probability density P. Here, T is used as a Lagrange multiplier to control the expected energy. With a high temperature, the problem space is dominated by the *entropy* term which makes the problem space become smooth so it is easy to move further. As the temperature is getting cooler, however, the problem space is gradually revealed as the landscape of the original cost function which limits the movement within the problem space. To avoid being trapped in local optima, people usually start with a high temperature and slowly decrease the temperature in the process of finding a solution.

SA relies on random sampling with the Monte Carlo method to estimate the expected solution, e.g. expected mapping in target dimension for MDS problem, so that it suffers from a long running time. Deterministic annealing (DA) [62, 63] can be thought of as an approximation algorithm of SA which tries to keep the merits of SA. The DA [62, 63] 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 the computational annealing process using Eq. (2.22), in which T decreases from high to low.

3

High Performance Multidimensional Scaling

3.1 Overview

Due to the innovative advancements in science and technology, the amount of data to be processed or analyzed is rapidly growing and it is already beyond the capacity of most commodity hardware we are using currently. To keep up with such fast development, study for data-intensive scientific data analyses [28] has been already emerging in recent years. It is a challenge for various computing research communities, such as high-performance computing, database, and machine learning and data mining communities, to learn how to deal with such large and high dimensional data in this data deluged era. Unless developed and implemented carefully to overcome such limits, techniques will face soon the limits of usability. Parallelism is not an optional technology any more but an essential factor for various data mining algorithms, including dimension reduction algorithms, by the result of the enormous size of the data to be dealt by those algorithms (especially since the data size keeps increasing).

Visualization of high-dimensional data in low-dimensions is an essential tool for exploratory data analysis, when people try to discover meaningful information which is concealed by the inherent complexity of the data, a characteristic which is mainly dependent on the high dimensionality of the data. This task is also

getting more difficult and challenged by the huge amount of the given data. In most data analysis with such large and high-dimensional dataset, we have observed that such a task is no more CPU bounded but rather memory bounded, in that any single process or machine cannot hold the whole data in its memory any longer.

In this chapter, I tackle this problem for developing a high performance visualization for large and high-dimensional data analysis by using distributed resources with parallel computation. For this purpose, we will show how we developed a well-known dimension-reduction-based visualization algorithm, named Multidimensional Scaling (MDS), in the distributed fashion so that one can utilize distributed memories and be able to process large and high dimensional datasets.

In this chapter, we introduce the details of our parallelized version of an MDS algorithm, called parallel SMACOF, in Section 3.2. The brief introduction of SMACOF [20, 21] can be found at Section 2.2 of Chapter 2. In the next, we show our performance results of our parallel version of MDS in various compute cluster settings, and we present the results of processing up to 100,000 data points in Section 3.3 followed by the summary of this chapter in Section 3.4.

3.2 High Performance Visualization

We have observed that processing a very large dataset is not only a *cpu-bounded* but also a *memory-bounded* computation, in that memory consumption is beyond the ability of a single process or even a single machine, and that it will take an unacceptable running time to run a large data set even if the required memory is available in a single machine. Thus, running machine learning algorithms to process a large dataset, including MDS discussed in this thesis, in a distributed fashion is crucial so that we can utilize multiple processes and distributed resources to handle very large data which usually will take a long time and even not fit in the memory of a single process or a compute node. The memory shortage problem becomes more obvious if the running OS is 32-bit which can handle at most 4GB virtual memory per process. To

Table 3.1: Main matrices used in SMACOF

Matrix	Size	Description
Δ	$N \times N$	Matrix for the given pairwise dissimilarity $[\delta_{ij}]$
$D(X)$	$N \times N$	Matrix for the pairwise Euclidean distance of mapped in target dimension $[d_{ij}]$
V	$N \times N$	Matrix defined by the value v_{ij} in (2.14)
$V^{\dagger}$	$N \times N$	Matrix for pseudo-inverse of V
$B(Z)$	$N \times N$	Matrix defined by the value b_{ij} in (2.16)
W	$N \times N$	Matrix for the weight of the dissimilarity $[w_{ij}]$
$X^{[k]}$	$N \times L$	Matrix for current L-dimensional configuration of N data points $x_i^{[k]} (i = 1, \ldots, N)$
$X^{[k-1]}$	$N \times L$	Matrix for previous L-dimensional configuration of N data points $x_i^{[k-1]} (i = 1, \ldots, N)$

process large data with efficiency, we have developed the parallel version of MDS by using a Message Passing Interface (MPI) fashion. In the following, we will discuss more details on how we decompose data using the MDS algorithm to fit in the memory limit of a single process or machine. We will also discuss how to implement an MDS algorithm, called SMACOF, by using MPI primitives to get some computational benefits of parallel computing.

3.2.1 Parallel SMACOF

Table 3.1 describes frequently used matrices in the SMACOF algorithm. As shown in Table 3.1, the 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 a critical problem when we deal with a large data set, such as hundreds of 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, the SMACOF algorithm generally needs six $N \times N$ matrices as described in Table 3.1, 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 3.1 and some required

temporary buffers.

If the weight is uniform ($w_{ij} = 1, \forall i, j$), we can use only four constants for representing $N \times N$ V and $V^{\dagger}$ matrices in order to saving memory space. We, however, still need at least three $N \times N$ matrices, i.e. $D(X)$, Δ, and $B(X)$, which requires 240 GB memory for the above case, which is still an unfeasible amount of memory for a typical computer. That is why we have to implement a 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 memory distribution but also for computational distribution, since parallelization implicitly benefits 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 a 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 the full memory requirements of SMACOF algorithm. Figure 3.1 illustrates how we decompose each $N \times N$ matrix with 6 processes and $m = 2, n = 3$. Without a loss of generality, we assume $N\%m = N\%n = 0$ in Figure 3.1.

A process $P_k, 0 \leq k < p$ (sometimes, we will use P_{ij} for matching M_{ij}) is assigned to one rectangular block M_{ij} with respect to the simple block assignment equation in (3.1):

$$k = i \times n + j \tag{3.1}$$

where $0 \leq i < m, 0 \leq j < n$. For $N \times N$ matrices, such as $\Delta, V^{\dagger}, B(X^{[k]})$, and so on, each block M_{ij} is assigned to the corresponding process P_{ij}, and for $X^{[k]}$ and $X^{[k-1]}$ matrices, $N \times L$ matrices where L is the target

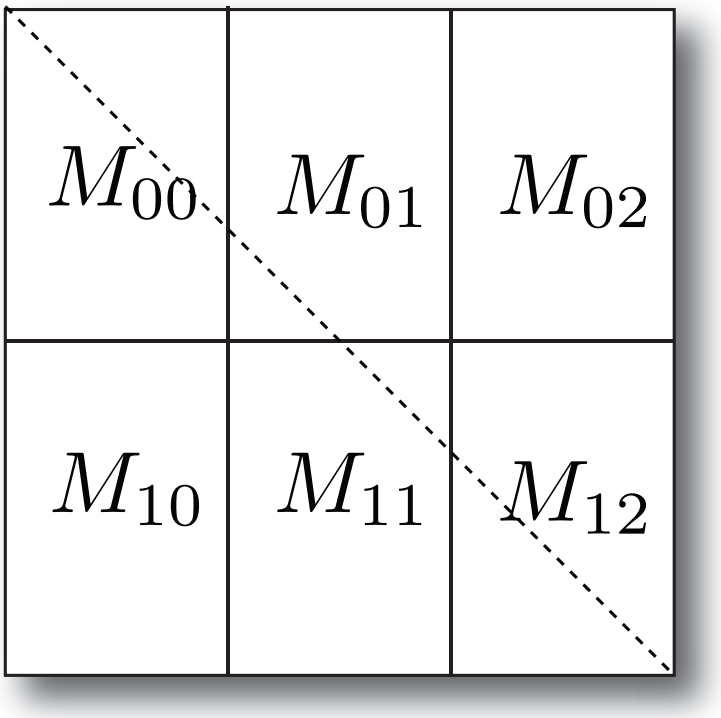


Figure 3.1: An example of an $N \times N$ matrix decomposition of parallel SMACOF with 6 processes and 2×3 block decomposition. Dashed line represents where diagonal elements are.

dimension, each process has a full $N \times L$ matrix because these matrices have a relatively smaller size, and this results in reducing the number of additional message passing routine calls. By scattering decomposed blocks throughout the distributed memory, we are now able to run SMACOF with as huge a data set as the distributed memory will allow concerning the cost of message passing overheads and a complicated implementation.

Although we assume $N\%m = N\%n = 0$ in Figure 3.1, there is always the possibility that $N\%m \neq 0$ or $N\%n \neq 0$. In order to achieve a high load balance under the $N\%m \neq 0$ or $N\%n \neq 0$ cases, we use a simple `modular` operation to allocate blocks to each process with at most ONE row or column difference between them. The block assignment algorithm is illustrated in Algorithm 3.1.

At the iteration k in Algorithm 2.2, the application should acquire up-to-date information of the following matrices: Δ, $V^{\dagger}$, $B(X^{[k-1]})$, $X^{[k-1]}$, and $\sigma^{[k]}$, to implement Line 8 and Line 9 in Algorithm 2.2. One good

Algorithm 3.1 Pseudo-code for block row and column assignment for each process for high load balance.

```
Input: pNum, N, myRank
 1: if N%pNum = 0 then
 2:    nRows = N / pNum;
 3: else
 4:    if myRank ≥ (N%pNum) then
 5:       nRows = N / pNum;
 6:    else
 7:       nRows = N / pNum + 1;
 8:    end if
 9: end if
10: return nRows;
```

feature of the SMACOF algorithm is that some of matrices are invariable, i.e. Δ and $V^{\dagger}$, through the iteration. On the other hand, $B(X^{[k-1]})$ and STRESS ($\sigma^{[k]}$) value keep changing at each iteration, since $X^{[k-1]}$ and $X^{[k]}$ are changed in every iteration. In addition, in order to update $B(X^{[k-1]})$ and the STRESS ($\sigma^{[k]}$) value in each iteration, we have to take the $N \times N$ matrices' information into account, so that related processes should communicate via MPI primitives to obtain the necessary information. Therefore, it is necessary to design message passing schemes to do parallelization for calculating the $B(X^{[k-1]})$ and STRESS ($\sigma^{[k]}$) values as well as the parallel matrix multiplication in Line 8 in Algorithm 2.2.

Computing the STRESS (Eq. (3.2)) can be implemented simply by a partial error sum of D_{ij} and Δ_{ij} followed by an `MPI_Allreduce`:

$$\sigma(X) \quad = \quad \sum_{i<j\leq N} w_{ij}(d_{ij}(X) - \delta_{ij})^2 \tag{3.2}$$

where $1 \leq i < j \leq N$ and w_{ij} is a weight value, so $w_{ij} \geq 0$. On the other hand, calculation of $B(X^{[k-1]})$, as shown at Eq. (2.16), and parallel matrix multiplication are not simple, especially for the case of $m \neq n$.

Figure 3.2 depicts how parallel matrix multiplication applies between an $N \times N$ matrix M and an $N \times L$ matrix X. Parallel matrix multiplication for SMACOF algorithm is implemented in a three-step process of message communication via MPI primitives. Block matrix multiplication of Figure 3.2 for acquiring C_i

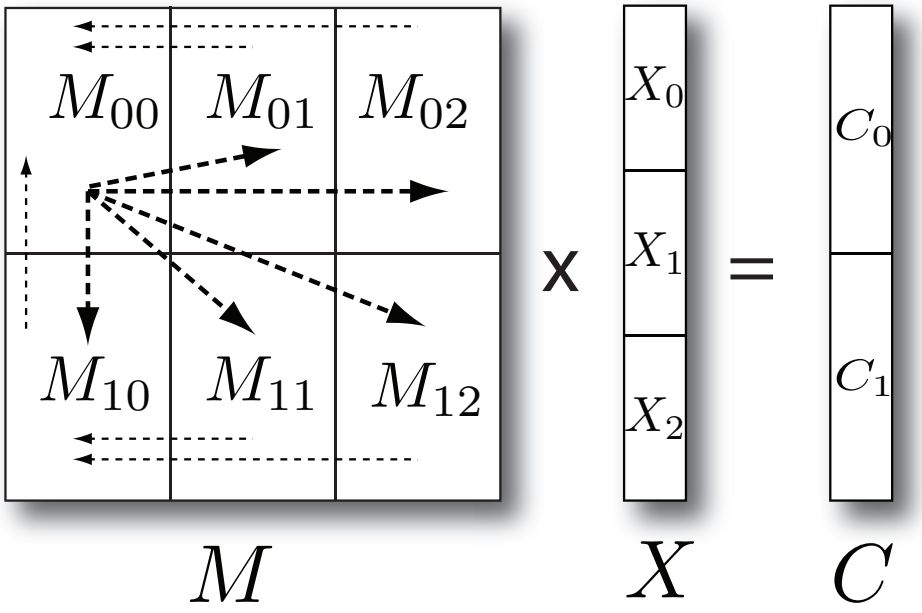


Figure 3.2: Parallel matrix multiplication of $N \times N$ matrix and $N \times L$ matrix based on the decomposition of Figure 3.1

$(i = 0, 1)$ can be written as follows:

$$C_i = \sum_{0 \le j < 3} M_{ij} \cdot X_j. \tag{3.3}$$

Since M_{ij} of $N \times N$ matrix is accessed only by the corresponding process P_{ij}, computing $M_{ij} \cdot X_j$ part is done by P_{ij}. Each computed sub-matrix by P_{ij}, which is $\frac{N}{2} \times L$ matrix for Figure 3.2, is sent to the process assigned M_{i0}, and the process assigned M_{i0}, say P_{i0}, sums the received sub-matrices to generate C_i by one collective MPI primitive call, such as `MPI_Reduce` with the `Addition` operation. Then P_{i0} sends C_i block to P_{00} by one collective MPI call, named `MPI_Gather`, as well. Note that we are able to use `MPI_Reduce` and `MPI_Gather` instead of `MPI_Send` and `MPI_Receive` by establishing row- and column-communicators for each process P_{ij}. `MPI_Reduce` is called under an established row-communicator, say $rowComm_i$ which is constructed by P_{ij} where $0 \le j < n$, and `MPI_Gather` is called under defined column-communicator

Algorithm 3.2 Pseudo-code for distributed parallel matrix multiplication in parallel SMACOF algorithm

Input: M_{ij}, X

/* $m =$ `Row Blocks`, $n =$ `Column Blocks` */
/* $i =$ `Rank-In-Row`, $j =$ `Rank-In-Column` */
/* $rowComm_i$: `Row Communicator` of row i, $rowComm_i \in P_{i0}, P_{i1}, P_{i2}, \ldots, P_{i(n-1)}$ */
/* $colComm_0$: `Column Communicator` of column 0, $colComm_0 \in P_{i0}$ where $0 \leq i < n$ */
$T_{ij} = M_{ij} \cdot X_j$
if $j \neq 0$ **then**
    /* Assume `MPI_Reduce` is defined as `MPI_Reduce(data, operation, root)` */
    Send T_{ij} to P_{i0} by calling `MPI_Reduce` (T_{ij}, `Addition`, P_{i0}).
else
    Generate C_i = `MPI_Reduce` (T_{i0}, `Addition`, P_{i0}).
end if
if $i == 0$ and $j == 0$ **then**
    /* Assume `MPI_Gather` is defined as `MPI_Gather(data, root)` */
    Gather C_i where $i = 0, \ldots, m-1$ by calling `MPI_Gather` (C_0, P_{00})
    Combine C with C_i where $0 \leq i < m$
    Broadcast C to all processes
else if $j == 0$ **then**
    Send C_i to P_{00} by calling `MPI_Gather`(C_i, P_{00})
    Receive C Broadcasted by P_{00}
else
    Receive C Broadcasted by P_{00}
end if

of P_{i0}, say $colComm_0$ whose members are P_{i0} where $0 \leq i < m$. Finally, P_{00} combines the gathered sub-matrix blocks C_i, where $0 \leq i < m$, to construct $N \times L$ matrix C, and broadcasts it to all other processes by `MPI_Broadcast` call.

Each arrow in Figure 3.2 represents message passing direction. Thin dashed arrow lines describes message passing of $\frac{N}{2} \times L$ sub-matrices by either `MPI_Reduce` or `MPI_Gather`, and message passing of matrix C by `MPI_Broadcast` is represented by thick dashed arrow lines. The pseudo code for parallel matrix multiplication in SMACOF algorithm is in Algorithm 3.2

For the purpose of computing $B(X^{[k-1]})$ in parallel, whose elements b_{ij} is defined in (2.17), the message passing mechanism in Figure 3.3 should be applied under a 2×3 block decomposition, as in Figure 3.1. Since $b_{ss} = -\sum_{s \neq j} b_{sj}$, a process P_{ij} assigned to B_{ij} should communicate a vector s_{ij}, whose element is the sum of

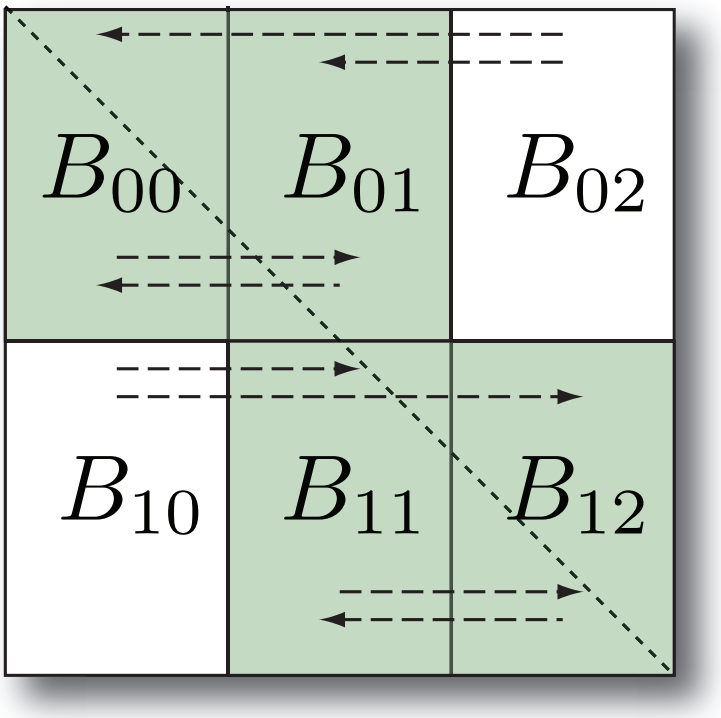


Figure 3.3: Calculation of $B(X^{[k-1]})$ matrix with regard to the decomposition of Figure 3.1.

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 Figure 3.3, 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* is a block which contains at least one diagonal element of the matrix $B(X^{[k]})$. Also, dashed arrow lines illustrate the message passing direction. The same as in parallel matrix multiplication, we use a collective call, i.e. `MPI_Allreduce` of row-communicator with `Addition` operation, to calculate row sums for the diagonal values of B instead of using pairwise communicate routines, such as `MPI_Send` and `MPI_Receive`. Algorithm 3.3 shows the pseudo-code of computing sub-block B_{ij} in process P_{ij} with MPI primitives.

Algorithm 3.3 Pseudo-code for calculating assigned sub-matrix B_{ij} defined in (2.17) for distributed-memory decomposition in parallel SMACOF algorithm

Input: M_{ij}, X

1: /* $m =$ `Row Blocks`, $n =$ `Column Blocks` */
2: /* $i =$ `Rank-In-Row`, $j =$ `Rank-In-Column` */
3: /* We assume that sub-matrix B_{ij} is assigned to process P_{ij} */
4: Find diagonal blocks in the same row (row i)
5: **if** $B_{ij} \notin$ diagonal blocks **then**
6: compute elements b_{st} of B_{ij}
7: Send a vector s_{ij}, whose element is the sum of corresponding rows, to P_{ik}, where $B_{ik} \in$ diagonal blocks. For simple and efficient implementation, we use `MPI_Allreduce` call for this.
8: **else**
9: compute elements b_{st} of B_{ij}, where $s \neq t$
10: Receive a vector s_{ik}, whose element is the sum of corresponding rows, where $k = 1, \ldots, n$ from other processes in the same block-row, and sum them to compute a row-sum vector by `MPI_Allreduce` call.
11: Compute b_{ss} elements based on the row sums.
12: **end if**

3.3 Performance Analysis of the Parallel SMACOF

For the performance analysis of parallel SMACOF discussed in this chapter, we have applied our parallel SMACOF algorithm for high-dimensional data visualization in low-dimension to the dataset obtained from the PubChem database[1], which is an NIH-funded repository for over 60 million chemical molecules. It provides their chemical structure fingerprints and biological activities, for the purpose of chemical information mining and exploration. Among 60 Million PubChem dataset, in this chapter we have used 100,000 randomly selected chemical subsets and all of them have a 166-long binary value as a fingerprint, which corresponds to the maximum input of 100,000 data points having 166 dimensions.

In the following, we will show the performance results of our parallel SMACOF implementation with respect to 6,400, 12,800, 50,000 and 100,000 data points having 166 dimensions, represented as 6400, 12800, 50K, and 100K datasets, respectively.

In addition to the PubChem dataset, we also use a biological sequence dataset for our performance test.

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

Table 3.2: 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
L1 (data) Cache per core	64 KB	32 KB
L2 Cache per core	512 KB	1.5 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

The biological sequence dataset contains 30,000 biological sequence data with respect to the metagenomics study based on pairwise distance matrix. Using these data as inputs, we have performed our experiments on our two decent compute clusters as summarized in Table 3.2.

Since we focus on analyzing the parallel runtime of the parallel SMACOF implementation but not mapping quality in this chapter, every experiment in this chapter is finished after 100 iterations without regard to the stop condition. In this way, we can measure parallel runtime performance with the same number of iterations for each data with different experimental environments.

3.3.1 Performance Analysis of the Block Decomposition

Figure 3.4-(a) and (c) show the overall elapsed time comparisons for the 6400 and 12800 PubChem data sets with respect to how to decompose the given $N \times N$ matrices with 32 cores in Cluster-I and Cluster-II. Also, Figure 3.4-(b) and (d) illustrate the partial runtime related to the calculation of $B(X)$ and the calculation of $D(X)$ of 6400 and 12800 PubChem data sets. An interesting characteristic of Figure 3.4-(a) and (c) is that matrix data decomposition does not much affect the execution runtime for a small data set (here 6400 points, in Figure 3.4-(a)), but for a large data set (here 12800 points, in Figure 3.4-(c)), row-based decomposition,

such as $p \times 1$, is severely worse in performance compared to other data decompositions. Figure 3.4-(c) and (d) describe that the overall performance with respect to data decomposition is highly connected to the calculation of the distance matrix runtime.

Also, Figure 3.5-(a) and (c) show the overall elapsed time comparisons for the 6400 and 12800 PubChem data sets with respect to how to decompose the given $N \times N$ matrices with 64 cores in Cluster-I and Cluster-II. Figure 3.5-(b) and (d) illustrate the partial runtimes related to the calculation of $B(X)$ and the calculation of $D(X)$ of 6400 and 12800 PubChem data sets, the same as Figure 3.4. Similar to Figure 3.4, the data decomposition does not make a substantial difference in the overall runtime of the parallel SMACOF with a small data set. However, row-based decomposition, in this case a 64×1 block decomposition, takes much longer for running time than the other decompositions, when we run the parallel SMACOF with the 12800 points data set. If we compare Figure 3.5-(c) with Figure 3.5-(d), we can easily find that the overall performance with respect to data decomposition is mostly affected by the calculation of the distance matrix runtime for the 64 core experiment.

The performance of overall elapsed time and partial runtimes of the 6400 and 12800 Pubchem data sets based on different decompositions of the given $N \times N$ matrices with 128 cores are experimented in only the Cluster-II system in Table 3.2. Those performance plots are shown in Figure 3.6. As shown in Figure 3.4 and Figure 3.5, the data decomposition does not have a considerable impact on the performance of the parallel SMACOF with a small data set but it does have a significant influence on that performance with a larger data set.

The main reason for the above data decomposition experimental results is the cache line effect that affects cache reusability, and generally balanced block decomposition shows better cache reusability so that it occurs with less cache misses than the skewed decompositions [6, 58]. In the parallel implementation of the SMACOF algorithm, two main components actually access data multiple times so that will be affected by cache reusability. One is the $[N \times N] \cdot [N \times D]$ matrix multiplication part. Since we implement the matrix

multiplication part based on the block matrix multiplication method with a 64×64 block for the purpose of better cache utilization, the runtime of matrix multiplication parts is almost the same without regard to data decomposition.

However, the distance matrix updating part is a tricky part. Since each entry of the distance matrix is accessed only once whenever the matrix is updated, it is not easy to think about the entries reusability. Although each entry of the distance matrix is accessed only once per each update, the new mapping points are accessed multiple times for calculation of the distance matrix. In addition, we update the distance matrix row-based direction for better locality. Thus, it is better for the number of columns to be small enough so that the coordinate values of each accessed mapping points for updating the assigned distance sub-matrix remain in the cache memory as much as is necessary.

Figure 3.4-(b),(d) through Figure 3.6-(b),(d) illustrate the cache reusability effect on 6400 points data and 12800 points data. For instance, for the row-based decomposition case, $p \times 1$ decomposition, each process is assigned an $N/p \times N$ block, i.e. 100×6400 data block for the cases of $N = 6400$ and $p = 64$. When $N = 6400$, the runtime of the distance matrix calculation part does not make much difference with respect to the data decomposition. We might consider that, if the number of columns of the assigned block is less than or equal to 6400, then cache utilization is no more harmful for the performance of the distance matrix calculation part of the parallel SMACOF. On the other hand, when $N = 12800$ which is doubled, the runtime of the distance matrix calculation part of row-based decomposition ($p \times 1$), and which is assigned a $12800/p \times 12800$ data block for each process, is much worse than the other data decomposition cases, as in sub-figure (d) of Figure 3.4 - Figure 3.6. For the other decomposition cases, such as $p/2 \times 2$ through $1 \times p$ data decomposition cases, the number of columns of the assigned block is less than or equal to 6400, when $N = 12800$, and the runtime performance of distance matrix calculation part of those cases are similar to each other and much less than the row-based data decomposition.

We have also investigated the runtime of the $B(X)$ calculation, since the message passing mechanism for

computing $B(X)$ is different based on data decomposition. Since the diagonal elements of $B(X)$ are the negative sum of elements in the corresponding rows, it is required to call `MPI_Allreduce` or `MPI_Reduce` MPI APIs for each row-communicator. Thus, the less number of column blocks means faster (or less MPI overhead) processes in computing $B(X)$, and even the row-based decomposition case does not need to call the MPI API for calculating $B(X)$. The effect of the different message passing mechanisms of $B(X)$ in regard to data decomposition is shown in sub-figure (b) and (d) of Figure 3.4 through Figure 3.6.

In terms of a system comparison between the two test systems in Table 3.2, Cluster-II performs better than Cluster-I in Figure 3.4 through Figure 3.6, although the clock speeds of the cores are similar to each other. There are two different factors between Cluster-I and Cluster-II in Table 3.2. We believe that those factors result in Cluster-II outperforming Cluster-I, i.e. L2 cache size and Networks. The L2 cache size per core is 3 times bigger in Cluster-II than Cluster-I, and Cluster-II connected by 20Gbps Infiniband but Cluster-I connected via 1Gbps Ethernet. Since SMACOF with large data is a memory-bound application, it is natural that the bigger cache size results in the faster running time.

3.3.2 Performance Analysis of the Efficiency and Scalability

In addition to data decomposition experiments, we measured the parallel scalability of parallel SMACOF in terms of the number of processes p. We investigated the scalability of parallel SMACOF by running with different number of processes, e.g. $p = 64$, 128, 192, and 256. On the basis of the above data decomposition experimental results, the balanced decomposition has been applied to this process scaling experiments. As p increases, the elapsed time should decrease, but linear performance improvement could not be achieved due to the parallel overhead.

We make use of the parallel efficiency value with respect to the number of parallel units for the purpose of measuriing scalability. Eq. (3.4) and Eq. (3.5) are the equations of overhead and efficiency calculations:

$$f = \frac{pT(p) - T(1)}{T(1)} \tag{3.4}$$

$$\varepsilon = \frac{1}{1+f} = \frac{T(1)}{pT(p)} \tag{3.5}$$

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. (3.4) and Eq. (3.5) can be replaced with Eq. (3.6) and Eq. (3.7) as follows:

$$f = \frac{\alpha T(p_1) - T(p_2)}{T(p_2)} \tag{3.6}$$

$$\varepsilon = \frac{1}{1+f} = \frac{T(p_2)}{\alpha T(p_1)} \tag{3.7}$$

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. (3.6) and Eq. (3.7) in order to calculate the overhead and corresponding efficiency, since it is impossible to run in a single machine for 50k and 100k data sets. Note that we used 16 computing nodes in Cluster-II (total memory size in 16 computing nodes is 768 GB) to perform the scaling experiment with a large data set, i.e. 50k and 100k PubChem data, since the SMACOF algorithm requires 480 GB memory for dealing with 100k data points, as we disscussed in Section 3.2.1, and Cluster-II can only perform that with more than 10 nodes.

The elapsed time of the parallel SMACOF with two large data sets, 50k and 100k, is shown in Figure 3.7-(a), and the corresponding relative efficiency of Figure 3.7-(a) is shown in Figure 3.7-(b). Note that both coordinates are log-scaled, in Figure 3.7-(a). As shown in Figure 3.7-(a), the parallel SMACOF achieved performance improvement as the number of parallel units (p) increases. However, the performance enhancement ratio (a.k.a. efficiency) is reduced as p increases, which is demonstrated in Figure 3.7-(b). The reason for

Table 3.3: Runtime Analysis of Parallel Matrix Multiplication part of parallel SMACOF with 50k data set in Cluster-II

#Procs	tMatMult	tMM_Computing	tMM_Overhead
64	668.8939	552.5348	115.9847
128	420.828	276.1851	144.2233
192	366.1	186.815	179.0401
256	328.2386	140.1671	187.8749

reducing efficiency is that the ratio of the message passing overhead over the assigned computation per each process is increased due to more message overhead and less computing portion per process as p increases, as shown in Table 3.3.

Table 3.3 is the result of the runtime analysis of the parallel matrix multiplication part of the proposed parallel SMACOF implementation which detached the time of the pure block matrix multiplication computation part and the time of the MPI message passing overhead part for parallel matrix multiplication, from the overall runtime of the parallel matrix multiplication part of the parallel SMACOF implementation. Note that **#Procs**, **tMatMult**, **tMM_Computing**, and **tMM_Overhead** represent the number of processes (parallel units), the overall runtime of the parallel matrix multiplication part, the time of the pure block matrix multiplication computation part, and the time of the MPI message passing overhead part for parallel matrix multiplication, respectively.

Theoretically, the **tMM_Computing** portion should be negatively linear with respect to the number of parallel units, if the number of points is the same and the load balance is achieved. Also, the **tMM_Overhead** portion should be increased as the number of parallel units is increased, if the number of points is the same. More specifically, if `MPI_Bcast` is implemented as one of the classical algorithms, such as a binomial tree or a binary tree algorithm [55], in MPI.NET library, then the **tMM_Overhead** portion will follow somewhat $\mathcal{O}(\lceil \lg p \rceil)$ with respect to the number of parallel units (p), since the `MPI_Bcast` routine in Algorithm 3.2 could be the most time consuming MPI method among the MPI routines of parallel matrix multiplication due in part to the large message size and the maximum number of communication participants.

Figure 3.8 illustrates the efficiency (calculated by Eq. (3.7)) of **tMatMult** and **tMM_Computing** in Table 3.3 with respect to the number of processes. As shown in Figure 3.8, the pure block matrix multiplication part shows very high efficiency, which is almost ONE. In other words, the pure block matrix multiplication part of the parallel SMACOF implementation achieves linear speed-up as we expected. Based on the efficiency measurement of **tMM_Computing** in Figure 3.8, we could conclude that the proposed parallel SMACOF implementation achieved good enough load balance and the major component of the decrease of the efficiency is the compulsary MPI overhead for implementing parallelism. By contrast, the efficiency of the overall runtime of the parallel matrix multiplication part is decreased to around 0.5, as we expected based on Table 3.3.

We also compare the measured MPI overhead of the parallel matrix multiplication (**tMM_Overhead**) in Table 3.3 with the estimation of the MPI overhead with respect to the number of processes. The MPI overhead is estimated based on the assumption that `MPI_Bcast` is implemented by a binomial tree or a binary tree algorithm, so that the runtime of `MPI_Bcast` is in $\mathcal{O}(\lceil \lg(p) \rceil)$ with respect to the number of parallel units (p). The result is described in Figure 3.9. In Figure 3.9, it is shown that the measured MPI overhead of the parallel matrix multiplication part has a similar shape with estimation overhead. We could conclude that the measured MPI overhead of the parallel matrix multiplication part takes the expected amount of time.

In addition to the experiment with pubChem data, which is represented by a vector format, we also experimented on the proposed algorithm with other real data sets, which contains 30,000 biological sequence data with respect to the metagenomics study (hereafter MC30000 data set). Although it is hard to present a biological sequence in a feature vector, researchers can calculate a dissimilarity value between two different sequences by using some pairwise sequence alignment algorithms, like Smith Waterman - Gotoh (*SW-G*) algorithm [30, 65] which we used here.

Figure 3.10 shows: (a) the runtime; and (b) the efficiency of the parallel SMACOF for the MC30000

data in Cluster-I and Cluster-II in terms of the number of processes. We tested it with 32, 64, 96, and 128 processes for Cluster-I, and experimented on it with more processes, i.e. 160, 192, 224, and 256 processes, for Cluster-II. Both (a) and (b) sub-figure of Figure 3.10 show similar tendencies to the corresponding sub-figure of Figure 3.7. If we compare Figure 3.10 to Figure 3.7, we can see that the efficiency of the parallel SMACOF for MC30000 data is generally lower than that of the parallel SMACOF for the 50k and 100k pubChem data sets.

3.4 Summary

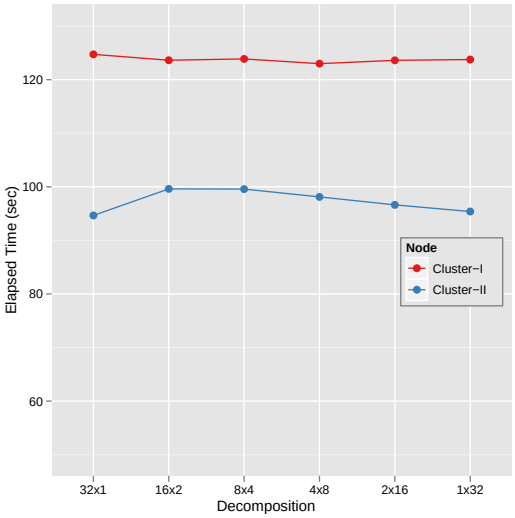
In this chapter, I have described a well-known dimension reduction algorithm, called MDS (SMACOF), and I have discussed how to utilize the algorithm for a huge data set. The main issues involved in dealing with a large amount of data points are not only lots of computations but also huge memory requirements. As we described in Section 3.2.1, it takes 480 GB of memory to run the SMACOF algorithm with 100,000 data points. Parallelization via the traditional MPI approach in order to utilize the distributed memory computing system, which can support much more computing power and extend the accessible memory size, is proposed as a solution for the amendment of the computation and memory shortage so as to be able to treat large data with SMACOF.

As we discussed in the performance analysis, the data decomposition structure is important to maximize the performance of the parallelized algorithm since it affects message passing routines and the message passing overhead as well as the cache-line effect. We look at overall elapsed time of the parallel SMACOF based on data decomposition as well as sub-routine runtimes, such as calculation of BofZ matrix ($B(X)$) and distance matrix ($D(X)$). The cache reusability affects the performance of updating the distance matrix of the newly generated mappings with respect to the data decomposition if we run a large data set. For a larger data set, row-based decomposition shows much longer runtime than others for the distance matrix calculation, and

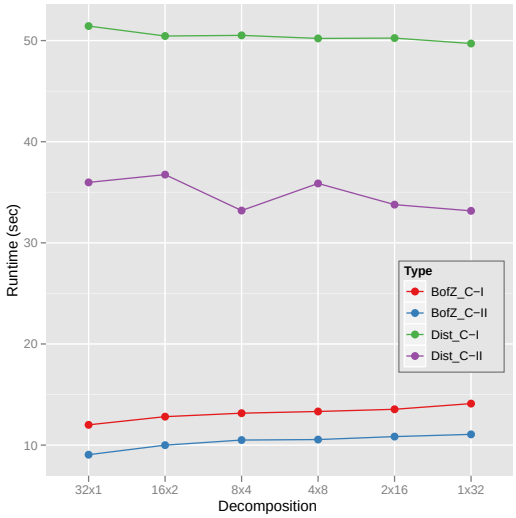
it influences the overall runtime as well. For the calculation of the BofZ matrix, the less column blocks case shows the better performance due to required row-based communication. From the above analysis, balanced data decomposition ($m \times n$) is generally better than skewed decomposition ($p \times 1$ or $1 \times p$) for the parallel MDS algorithm.

In addition to data decomposition analysis, we also analyzed the efficiency and the scalability of the parallel SMACOF. Although the efficiency of the parallel SMACOF is decreased by increasing the number of processes due to the increase of overhead and the decrease of pure parallel computing time, the efficiency is still good enough for a certain degree of parallelism. For instance, the efficiency is still over 70% with the 100k data set with 256 parallelism. Based on the fact that the **tMM_Computing** in Table 3.3 achieved almost linear speedup as in Figure 3.8, it is shown that the parallel SMACOF implementation deals with the load balance issue very well and the inevitable message passing overhead for parallelism is the main factor of the reduction of the efficiency.

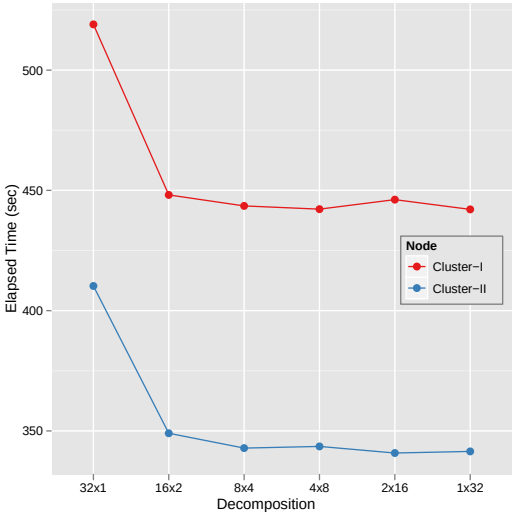
There are important problems for which the data set sizes are too large for even our parallel algorithms to be practical. Because of this, I developed interpolation approaches for the MDS algorithm, which could be synergied by the proposed parallel SMACOF implementation. Here we run normal MDS (or parallel MDS) with a (random) subset of the dataset (called *sample data*), and the dimension reduction of the remaining points are interpolated based on the pre-mapped mapping position of the sample data. The detail of the interpolation approach [7] is reported in Chapter 4.



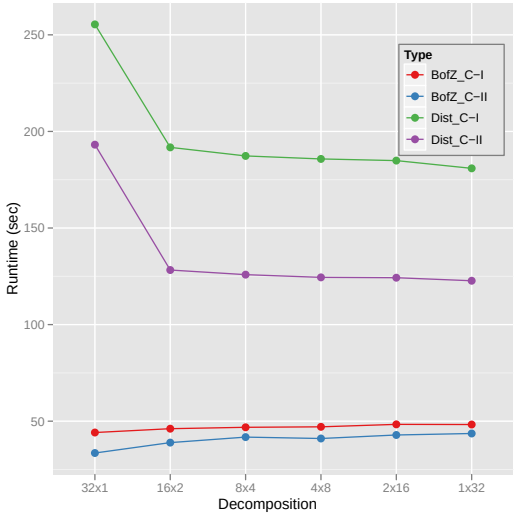
(a) 6400 with 32 cores



(b) partial run of 6400 with 32 cores

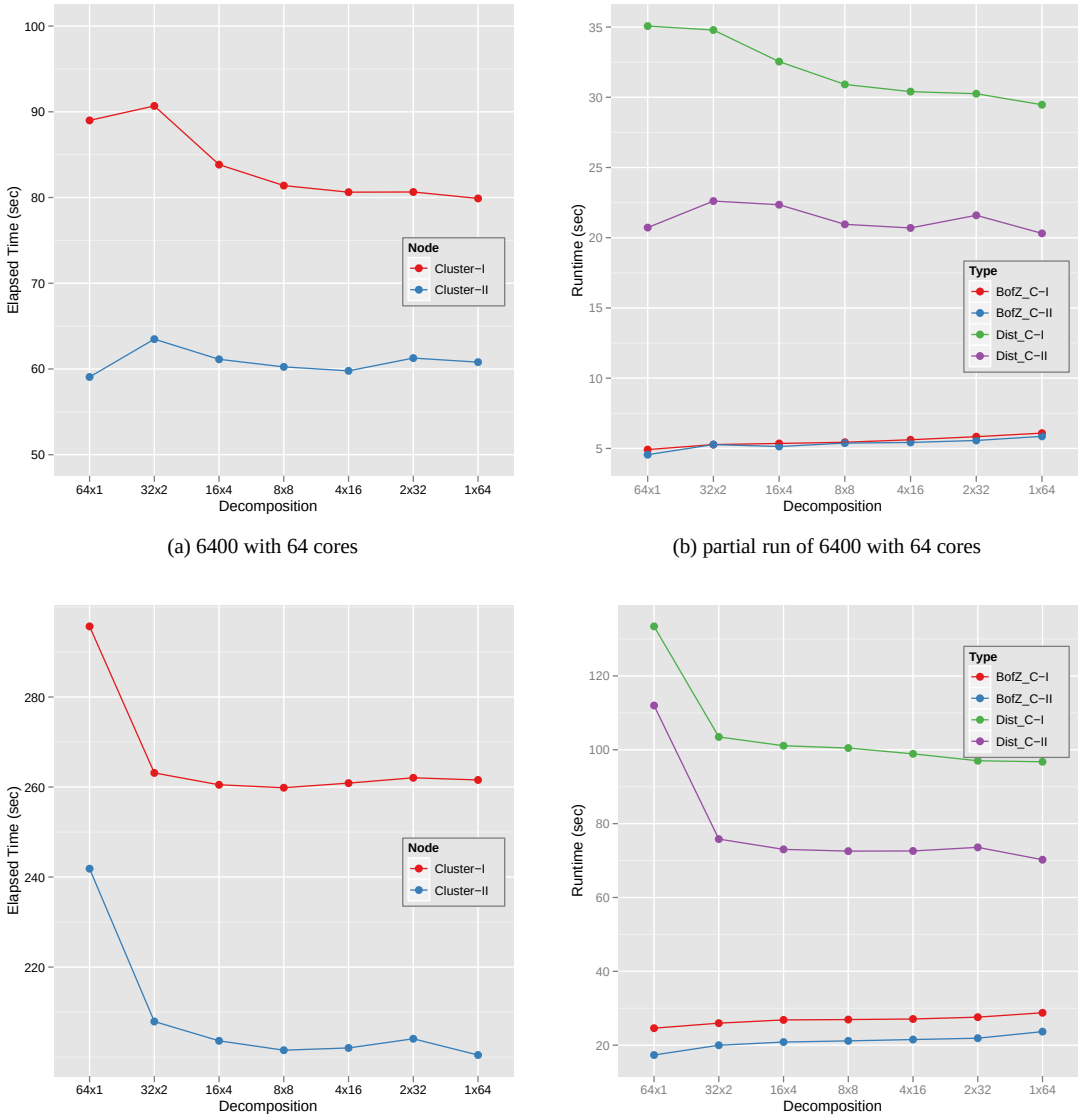


(c) 12800 with 32 cores



(d) partial run of 12800 with 32 cores

Figure 3.4: Overall Runtime and partial runtime of parallel SMACOF for 6400 and 12800 PubChem data with 32 cores in Cluster-I and Cluster-II w.r.t. data decomposition of $N \times N$ matrices.



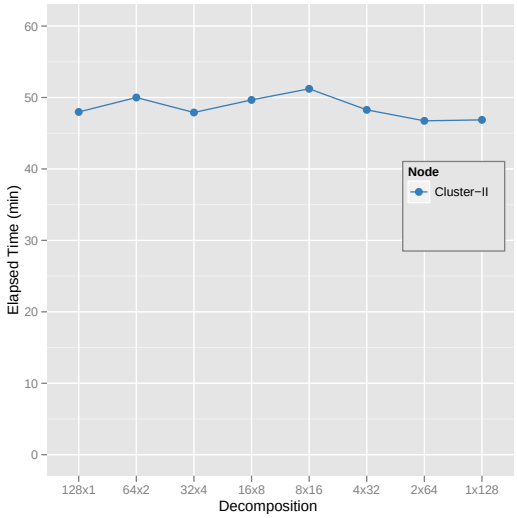
(a) 6400 with 64 cores

(b) partial run of 6400 with 64 cores

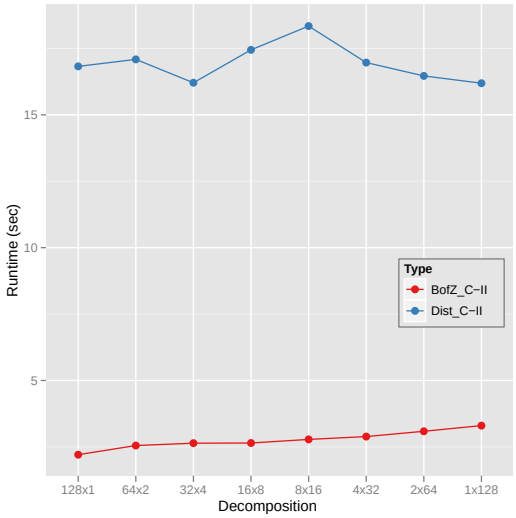
(c) 12800 with 64 cores

(d) partial run of 12800 with 64 cores

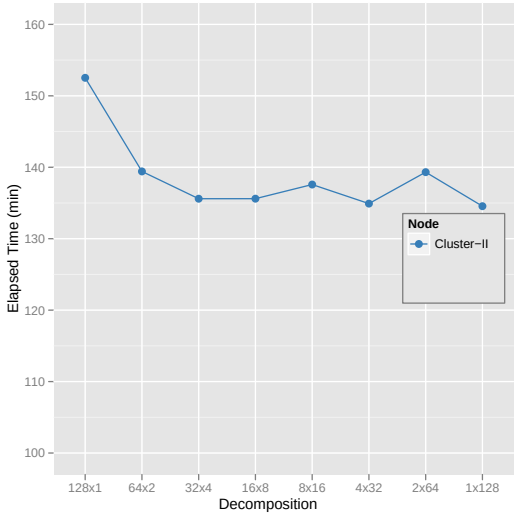
Figure 3.5: Overall Runtime and partial runtime of parallel SMACOF for 6400 and 12800 PubChem data with 64 cores in Cluster-I and Cluster-II w.r.t. data decomposition of $N \times N$ matrices.



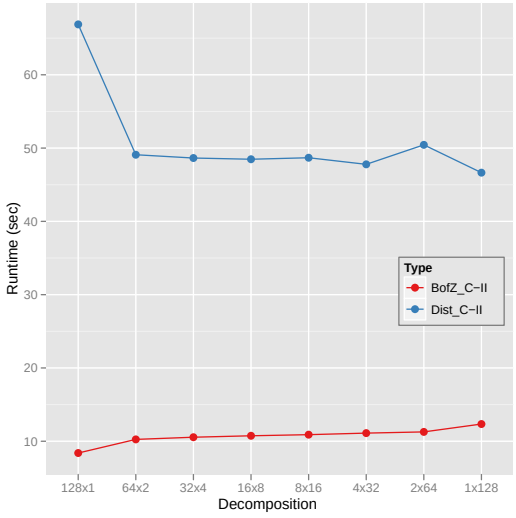
(a) 6400 with 128 cores



(b) partial run of 6400 with 128 cores

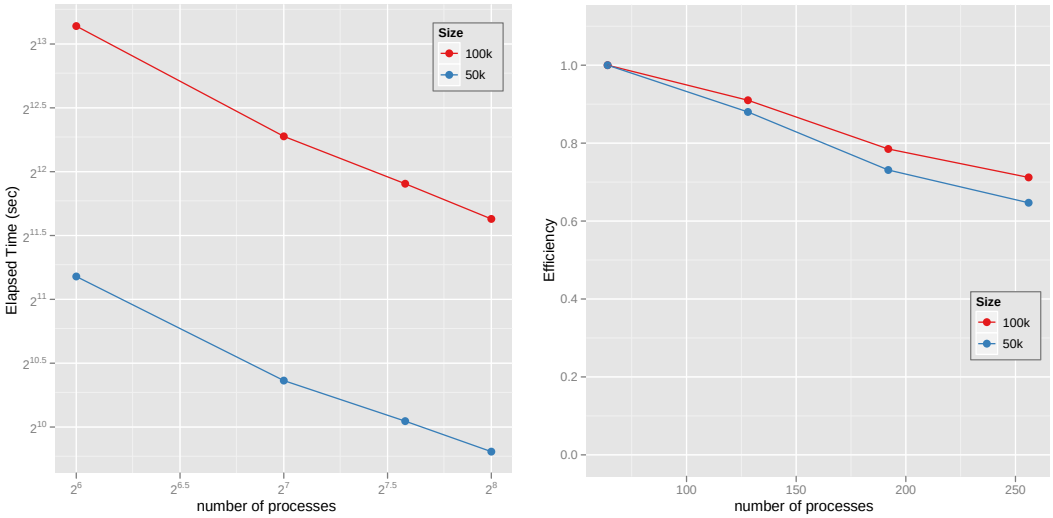


(c) 12800 with 128 cores



(d) partial run of 12800 with 128 cores

Figure 3.6: Overall Runtime and partial runtime of parallel SMACOF for 6400 and 12800 PubChem data with 128 cores in Cluster-II w.r.t. data decomposition of $N \times N$ matrices.



(a) large data runtime

(b) large data efficiency

Figure 3.7: Performance of parallel SMACOF for 50K and 100K PubChem data in Cluster-II w.r.t. the number of processes, i.e. 64, 128, 192, and 256 processes (cores). (a) shows runtime and efficiency is shown at (b). 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 for (a).

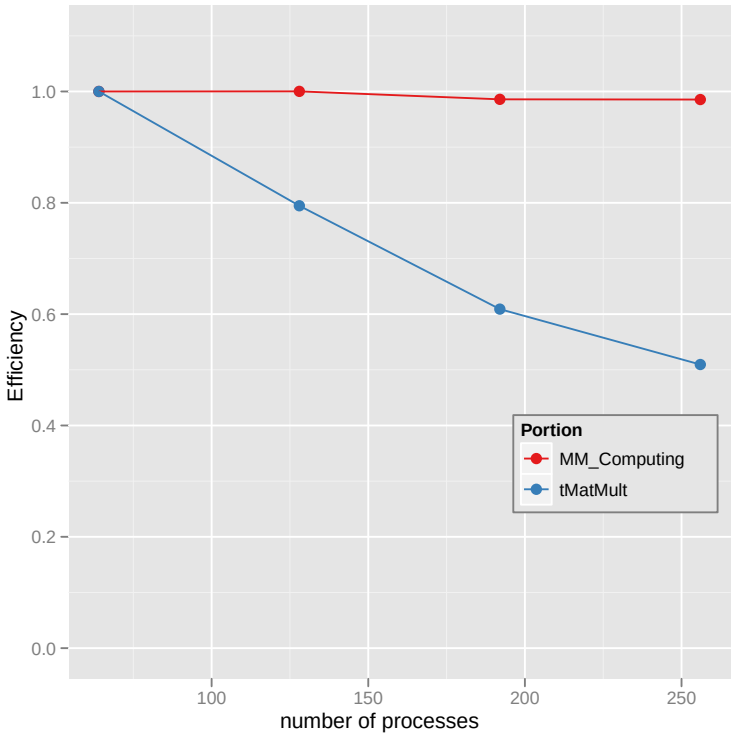


Figure 3.8: Efficiency of **tMatMult** and **tMM_Computing** in Table 3.3 with respect to the number of processes.

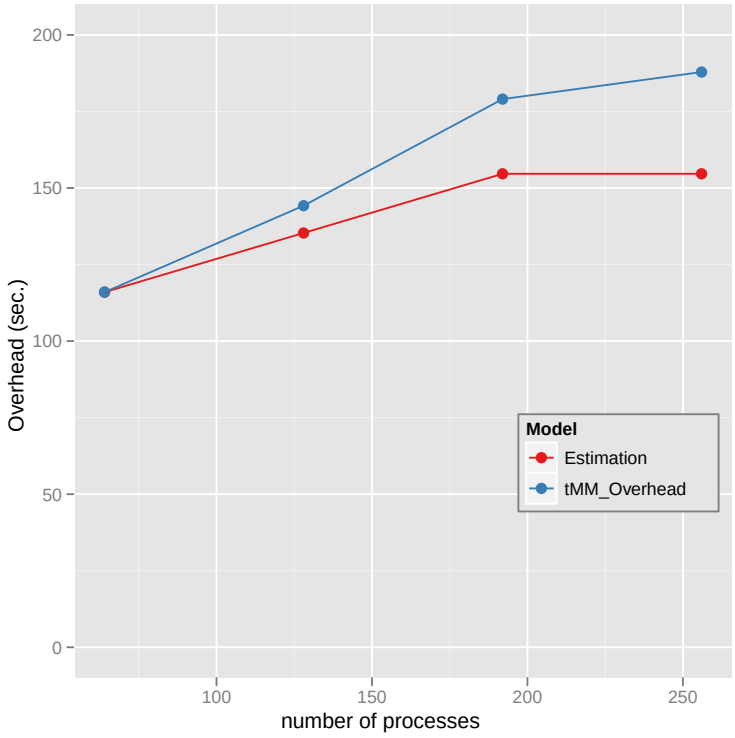
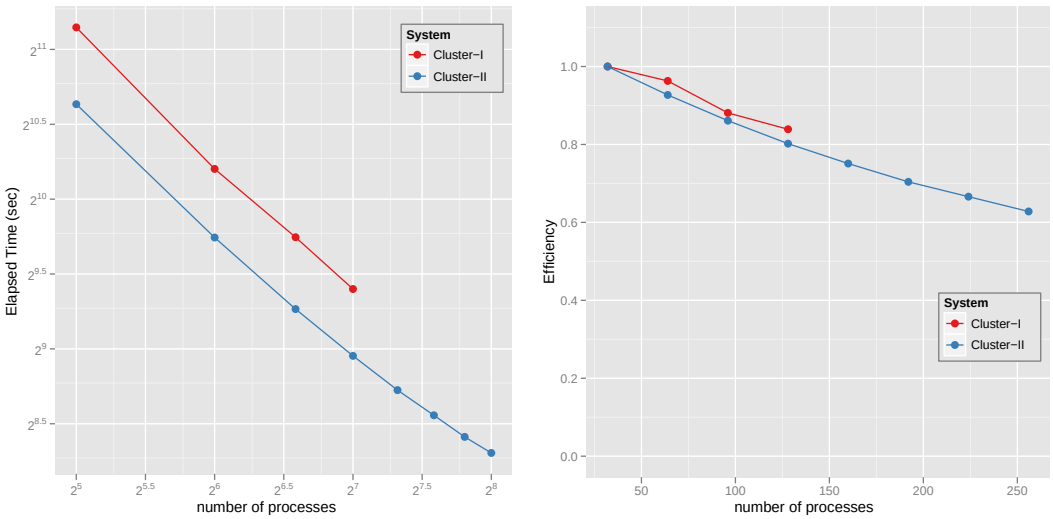


Figure 3.9: MPI Overhead of parallel matrix multiplication (**tMM_Overhead**) in Table 3.3 and the rough Estimation of the MPI overhead with respect to the number of processes.



(a) MC30000 runtime

(b) MC30000 efficiency

Figure 3.10: Performance of parallel SMACOF for MC 30000 data in Cluster-I and Cluster-II w.r.t. the number of processes, i.e. 32, 64, 96, and 128 processes for Cluster-I and Cluster-II, and extended to 160, 192, 224, and 256 processes for Cluster-II. (a) shows runtime and efficiency is shown at (b). 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 for (a).

4

Interpolation Approach for Multidimensional Scaling

4.1 Overview

Due to the advancements in science and technology over the last several decades, every scientific and technical field has generated a huge amount of data as time has passed in the world. We are really in the era of data deluge. In reflecting on the data deluge era, data-intensive scientific computing [28] has emerged in the scientific computing fields and it has been attracting more by many people. To analyze those incredible amount of data, many data mining and machine learning algorithms have been developed. Among the many data mining and machine learning algorithms that have been invented, we focus on dimension reduction algorithms, which reduce data dimensionality from original high dimension to target dimension, in this chapter.

Among the many dimension reduction algorithms which exist, such as principle component analysis (PCA), generative topographic mapping (GTM) [11, 12], self-organizing map (SOM) [43], multidimensional

scaling (MDS) [13, 45], I have worked on MDS for this thesis. Previously, we parallelize the MDS algorithm to utilize multicore clusters and to increase the computational capability with minimal overhead for the purpose of investigating large data, such as 100k data [16]. However, parallelization of an MDS algorithm, 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 utilizes distributed memory environments, such as clusters, for acquiring more memory and computational resources. In this chapter, we try to solve the memory-bound problem by interpolation based on pre-configured mappings of the sample data for the MDS algorithm, so that we can provide configuration of millions of points in the target space.

This chapter is organized as follows. First we will briefly discuss existing methods of *out-of-sample* problem in various dimension reduction algorithms in Section 4.2. Then, the proposed interpolation method and how to parallelize it are described in Section 4.3. The quality comparison between interpolated results and the full MDS running results and parallel performance evaluation of those algorithms are shown in Section 4.4, followed by the summary of this chapter in Section 4.5.

4.2 Related Work

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

In a sensor network localization field, when there are only a subset of pairwise distances between sensors and a subset of anchor locations are available, people try to find out the locations of the remaining sensors.

For instance, the semi-definite programming relaxation approaches and its extended approaches has been proposed to solve this issue [74]. [10] and [70] proposed out-of-sample extension for the classical multidimensional scaling (CMDS) [68], which is based on spectral decomposition of a symmetric positive semidefinite matrix (or the approximation of positive semidefinite matrix), and the embeddings in the configured space are represented in terms of eigenvalues and eigenvectors of it. [10] projected the new point x onto the principal components, and [70] extends the CMDS algorithm itself to the out-of-sample problem. In [70], the authors describe how to embed one point between the embeddings of the original n objects through modification of the original CMDS equations, which preserves the mappings of the original n objects, with $(n+1)\times(n+1)$ matrix A_2 instead of $n \times n$ matrix Δ_2, and extends to embedding a number of points simultaneously by using matrix operations. Recently, a multilevel force-based MDS algorithm was proposed as well [38].

In contrast to applying the out-of-sample problem to CMDS, I extend the out-of-sample problem to general MDS results with the STRESS criteria of Eq. (1.1) in Chapter 2, which finds embeddings of approximating to the distance (or dissimilarity) rather than the inner product as in CMDS, with an gradient descent optimization method, called iterative majorizing. The proposed iterative majorizing interpolation approach for the MDS problem will be explained in Section 4.3.

4.3 Majorizing Interpolation MDS

One of the main limitation of most MDS applications is that they require $\mathcal{O}(N^2)$ memory as well as $\mathcal{O}(N^2)$ computation. Thus, though it is possible to run them with a small data size without any trouble, it is impossible to execute them with a large number of data due to memory limitation; therefore, this challenge could be considered as being a memory-bound problem. For instance, Scaling by MAjorizing of COmplicated Function (SMACOF) [20, 21], a well-known MDS application via Expectation-Maximization (EM) [22] approach, uses six $N \times N$ matrices. If $N = 100,000$, then one $N \times N$ matrix of 8-byte double-precision numbers

requires 80 GB of main memory, so the algorithm needs to acquire at least 480 GB of memory to store these six $N \times N$ matrices. It is possible to run a parallel version of SMACOF with MPI in **Cluster-II** in Table 4.1 with $N = 100,000$. If the data size is increased only twice, however, then the SMACOF algorithm should have 1.92 TB of memory, which is bigger than the total memory of **Cluster-II** in Table 4.1 (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 the k nearest neighbor (k-NN) classification [19], but we approximate to a new mapping position of the new point based on the positions of k-NN, among pre-mapped subset data, instead of classifying it. For the purpose of deciding a new mapping position in relation to the k-NN positions, the iterative majorization method is applied as in the SMACOF [20, 21] algorithm. The details of mathematical majorization equations for the proposed out-of-sample MDS algorithm is shown below. The algorithm proposed in this chapter is called Majorizing Interpolation MDS (hereafter *MI-MDS*).

The proposed algorithm is implemented as follows. We are given N data in a high-dimensional space, say D-dimension, and proximity information ($\Delta = [\delta_{ij}]$) of those data as in Section 1.2. Among N data, the configuration of the n sample points in L-dimensional space, $x_1, \ldots, x_n \in \mathbb{R}^L$, called X, are already constructed by an MDS algorithm; here we use the SMACOF algorithm. Then, we select k nearest neighbors ($p_1, \ldots, p_k \in P$) of the given new point, among n pre-mapped points with respect to corresponding δ_{ix}, where x represents the new point. I use a linear search to find the k-nearest neighbors among n-sampled data, so that the complexity of finding the k-nearest neighbors is $\mathcal{O}(n)$ per one interpolated point (here x). Finally, the new mapping of the given new point $x \in \mathbb{R}^L$ is calculated based on the pre-mapped position of the selected k-NN and the corresponding proximity information δ_{ix}. The finding new mapping position is considered as a minimization problem of STRESS (4.1) as similar as normal MDS problem with m points, where $m = k + 1$.

However, only one point x is movable among m points, so we can simplify the STRESS equation (4.1) as follows (Eq. (4.2)), and we set $w_{ij} = 1$, for $\forall i, j$ in order to simplify.

$$\sigma(X) = \sum_{i<j\leq m} (d_{ij}(X) - \delta_{ij})^2 \tag{4.1}$$

$$= \mathscr{C} + \sum_{i=1}^{k} d_{ix}^2 - 2\sum_{i=1}^{k} \delta_{ix} d_{ix} \tag{4.2}$$

where δ_{ix} is the original dissimilarity value between p_i and x, d_{ix} is the Euclidean distance in L-dimension between p_i and x, and $\mathscr{C}$ is constant part. The second term of Eq. (4.2) can be deployed as following:

$$\sum_{i=1}^{k} d_{ix}^2 = \|x - p_1\|^2 + \cdots + \|x - p_k\|^2 \tag{4.3}$$

$$= k\|x\|^2 + \sum_{i=1}^{k} \|p_i\|^2 - 2x^t q \tag{4.4}$$

where $q^t = (\sum_{i=1}^{k} p_{i1}, \ldots, \sum_{i=1}^{k} p_{iL})$ and p_{ij} represents j-th element of p_i. In order to establish majorizing inequality, we apply *Cauchy-Schwarz* inequality to $-d_{ix}$ of the third term of Eq. (4.2). Please, refer to chapter 8 in [13] for details of how to apply the *Cauchy-Schwarz* inequality to $-d_{ix}$. Since $d_{ix} = \|p_i - x\|$, $-d_{ix}$ could have following inequality based on *Cauchy-Schwarz* inequality:

$$-d_{ix} \leq \frac{\sum_{a=1}^{L}(p_{ia} - x_a)(p_{ia} - z_a)}{d_{iz}} \tag{4.5}$$

$$= \frac{(p_i - x)^t (p_i - z)}{d_{iz}} \tag{4.6}$$

where $z^t = (z_i, \ldots, z_L)$ and $d_{iz} = \|p_i - z\|$. The equality in Eq. (4.5) occurs if x and z are equal. If we apply

Eq. (4.6) to the third term of Eq. (4.2), then we obtain

$$-\sum_{i=1}^{k}\delta_{ix}d_{ix} \leq -\sum_{i=1}^{k}\frac{\delta_{ix}}{d_{iz}}(p_i-x)^t(p_i-z) \tag{4.7}$$

$$= -x^t\sum_{i=1}^{k}\frac{\delta_{ix}}{d_{iz}}(z-p_i)+\mathscr{C}_\rho \tag{4.8}$$

where $\mathscr{C}_\rho$ is a constant. If Eq. (4.4) and Eq. (4.8) are applied to Eq. (4.2), then it could be like following:

$$\sigma(X) = \mathscr{C}+\sum_{i=1}^{k}d_{ix}^2-2\sum_{i=1}^{k}\delta_{ix}d_{ix} \tag{4.9}$$

$$\leq \mathscr{C}+k\|x\|^2-2x^tq+\sum_{i=1}^{k}\|p_i\|^2 - 2x^t\sum_{i=1}^{k}\frac{\delta_{ix}}{d_{iz}}(z-p_i)+\mathscr{C}_\rho \tag{4.10}$$

$$= \tau(x,z) \tag{4.11}$$

where both $\mathscr{C}$ and $\mathscr{C}_\rho$ are constants. In the Eq. (4.11), $\tau(x,z)$, a quadratic function of x, is a majorization function of the STRESS. Through setting the derivative of $\tau(x,z)$ equal to zero, we can obtain a minimum of it; that is

$$\nabla\tau(x,z) = 2kx-2q-2\sum_{i=1}^{k}\frac{\delta_{ix}}{d_{iz}}(z-p_i)=0 \tag{4.12}$$

$$x = \frac{q+\sum_{i=1}^{k}\frac{\delta_{ix}}{d_{iz}}(z-p_i)}{k} \tag{4.13}$$

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

neighbors that we selected.

The advantage of the iterative majorization algorithm is that it produces a series of mappings with non-increasing STRESS values as proceeds, which results in local minima. It is good enough to find local minima, since the proposed MI algorithm simplifies the complicated non-linear optimization problem as a small non-linear optimization problem, such as $k+1$ points non-linear optimization problem, where $k \ll N$. Finally, if we substitute z with $x^{[t-1]}$ in Eq. (4.13), then we generate an iterative majorizing equation like the following:

$$x^{[t]} = \frac{q + \sum_{i=1}^{k} \frac{\delta_{ix}}{d_{iz}}(x^{[t-1]} - p_i)}{k} \tag{4.14}$$

$$x^{[t]} = \overline{p} + \frac{1}{k}\sum_{i=1}^{k} \frac{\delta_{ix}}{d_{iz}}(x^{[t-1]} - p_i) \tag{4.15}$$

where $d_{iz} = \|p_i - x^{[t-1]}\|$ and $\overline{p}$ is the average of k-NN's mapping results. Eq. (4.15) is an iterative equation used to embed newly added point into target-dimensional space, based on pre-mapped positions of k-NN. The iteration stop condition is essentially the same as that of the SMACOF algorithm, which is

$$\Delta\sigma(S^{[t]}) = \sigma(S^{[t-1]}) - \sigma(S^{[t]}) < \varepsilon, \tag{4.16}$$

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

The time complexity of the proposed MI-MDS algorithm to find the mapping of one interpolated point is $\mathcal{O}(k)$ on the basis of Eq. (4.15), if we assume that the number of iterations of finding one interpolated mapping is very small. Since finding nearest neighbors takes $\mathcal{O}(n)$ and mapping via MI-MDS requires $\mathcal{O}(k)$ for one interpolated point, the overall time complexity to find mappings of overall out-of-sample points (N-n points) via the proposed MI-MDS algorithm is $\mathcal{O}(kn(N-n)) \approx \mathcal{O}(n(N-n))$, due to the fact that k is usually negligible compared to n or N.

Algorithm 4.1 Majorizing Interpolation (MI) algorithm

1: Find k-NN: find k nearest neighbors of x, $p_i \in P$ $i = 1,\ldots,k$ of the given new data based on original dissimilarity δ_{ix}.
2: Gather mapping results in target dimension of the k-NN.
3: Calculate $\overline{p}$, the average of pre-mapped results of $p_i \in P$.
4: Generate initial mapping of x, called $x^{[0]}$, either $\overline{p}$ or a random variation from $\overline{p}$ point.
5: Compute $\sigma(S^{[0]})$, where $S^{[0]} = P \cup \{x^{[0]}\}$.
6: **while** $t = 0$ or $(\Delta\sigma(S^{[t]}) > \varepsilon$ and $t \leq$ MAX_ITER) **do**
7: increase t by one.
8: Compute $x^{[t]}$ by Eq. (4.15).
9: Compute $\sigma(S^{[t]})$.
10: **end while**
11: **return** $x^{[t]}$;

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

The summary of the proposed MI algorithm for interpolation of a new data, say x, in relation to pre-mapping result of the sample data is described in Algorithm 4.1. Note that the algorithm uses $\overline{p}$ as an initial mapping of the new point $x^{[0]}$ unless initialization with $\overline{p}$ makes $d_{ix} = 0$, since the mapping is based on the k-NN. $\overline{p}$ makes $d_{ix} = 0$, if and only if all the mapping positions of the k-NNs are on the same position. If $\overline{p}$ makes $d_{ix} = 0$ $(i = 1,\ldots,k)$, then we generate a random variation from the $\overline{p}$ point with the average distance of δ_{ix} as an initial position of $x^{[0]}$.

4.3.1 Parallel MI-MDS Algorithm

Suppose that, among N points, the mapping results of n sample points in the target dimension, say L-dimension, are given so that we could use those pre-mapped results of n points via MI-MDS algorithm which is described above to embed the remaining points $(M = N - n)$. Though interpolation approach is much faster than full running MDS algorithm, i.e. $\mathcal{O}(Mn + n^2)$ vs. $\mathcal{O}(N^2)$, implementing parallel MI-MDS algorithm is

essential, since M can be still huge, like millions. In addition, most of clusters are now in forms of multicore-clusters after the invention of the multicore-chip, so we are using hybrid-model parallelism, which combine processes and threads together as used in [28, 57].

In contrast to the original MDS algorithm in which the mapping of a point is influenced by the other points, interpolated points are totally independent one another, except selected k-NN in the MI-MDS algorithm, and the independency of among interpolated points makes the MI-MDS algorithm to be pleasingly-parallel. In other words, there must be minimum communication overhead. Also, load-balance can be achieved by using modular calculation to assign interpolated points to each parallel unit, either between processes or between threads, as the number of assigned points are different at most one.

4.3.2 Parallel Pairwise Computation Method 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 of Eq. (1.1) 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 a distributed-memory system, such as cluster systems, to access only a subset of huge data which spread to over the clusters, so that each process needs to communicate each other for the purpose of accessing all the necessary data to compute STRESS.

In this section, we illustrate how to calculate symmetric pairwise computation efficiently in parallel with the case that only a subset of data is available for each process. In fact, general MDS algorithms utilize pairwise dissimilarity information, but suppose we are given N original vectors in D-dimension, $y_i, \ldots, y_N \in Y$ and $y_i \in \mathbb{R}^D$, instead of a given dissimilarity matrix, as PubChem finger print data that we used for our experiments. Thus, in order to calculate the distance in original D-dimension $\delta_{ij} = \|y_i - y_j\|$ in Eq. (1.1), it is necessary to communicate messages between each process to get the required original vector, say y_i

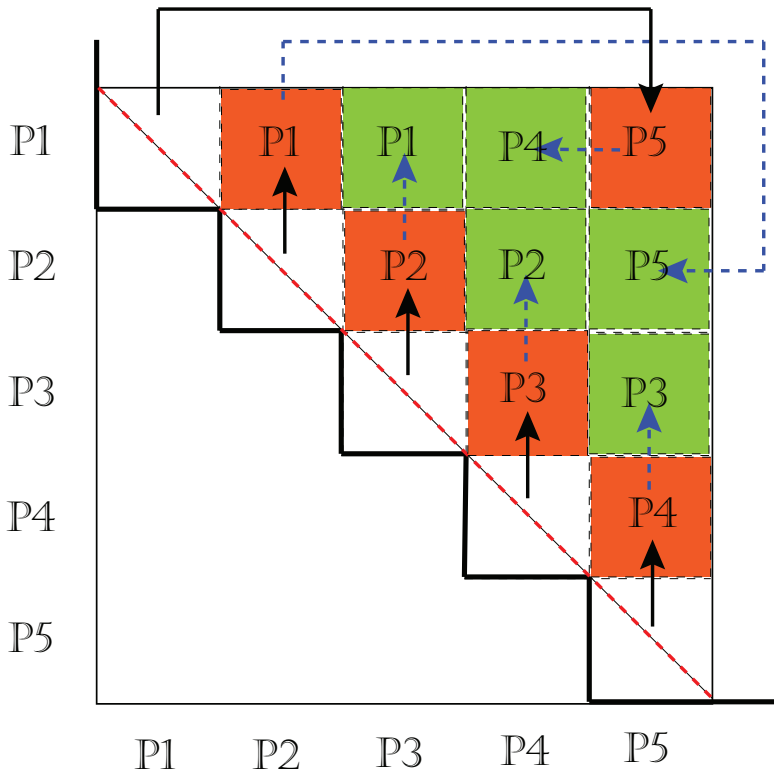


Figure 4.1: Message passing pattern and parallel symmetric pairwise computation for calculating STRESS value of whole mapping results.

and y_j. Here, we used the proposed pairwise computation method to measure the STRESS criteria of MDS problem in Eq. (1.1), but the proposed parallel pairwise computation method will be used efficiently for general parallel pairwise computation whose computing components are independent, such as generating a distance (or dissimilarity) matrix of all data, under the condition that each process can access only a subset of the required data.

Figure 4.1 describes the proposed scheme when the number of processes (p) is 5, odd numbers. The proposed scheme is an iterative two-step approach, (1) rolling and (2) computing, and the iteration number is $\lceil (1+\cdots+p-1)/p \rceil = \lceil (p-1)/2 \rceil$. Note that iteration ZERO is calculating the upper triangular part of the corresponding diagonal block, which does not requires message passing. After iteration ZERO is done, each process p_i sends the originally assigned data block to the previous process p_{i-1} and receives a data block

from the next process p_{i+1} in cyclic way. For instance, process p_0 sends its own block to process p_{p-1}, and receives a block from process p_1. This rolling message passing can be done using one single MPI primitive per process, `MPI_SENDRECV()`, which is efficient. After sending and receiving messages, each process performs currently available pairwise computing block with respect to receiving data and originally assigned block. In Figure 4.1, black solid arrows represent each message passings at iteration 1, and orange blocks with process ID are the calculated blocks by the corresponding named process at iteration 1. From iteration 2 to iteration $\lceil (p-1)/2 \rceil$, the above two-steps are done repeatedly and the only difference is nothing but sending a received data block instead of the originally assigned data block. The green blocks and dotted blue arrows show iteration 2 which is the last iteration for the case of $p = 5$.

Also, for the case that the number of processes is even, the above two-step scheme works in high efficiency. The only difference between the odd number case and the even number case is that two processes are assigned to one block at the last iteration of even number case, but not in an odd number case. Though two processes are assigned to a single block, it is easy to achieve load balance by dividing two sections of the block and assigning them to each process. Therefore, both odd number process and even number process cases are parallelized well using the above rolling-computing scheme, with minimal message passing overhead. The summary of the above parallel pairwise computation is shown in Algorithm 4.2.

4.4 Analysis of Experimental Results

To measure the quality and parallel performance of the proposed MDS interpolation (*MI-MDS*) approach discussed in this chapter, we have used 166-dimensional chemical dataset obtained from the PubChem project database[1], which is an NIH-funded repository for over 60 million chemical molecules and provides their chemical structures and biological activities, for the purpose of chemical information mining and exploration.

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

Algorithm 4.2 Parallel Pairwise Computation

```
1: input: Y = a subset of data;
2: input: p = the number of process;
3: rank ⇐ the rank of process;
4: sendTo ⇐ (rank − 1) mod p
5: recvFrom ⇐ (rank + 1) mod p
6: k ⇐ 0;
7: Compute upper triangle in the diagonal blocks in Figure 4.1;
8: MAX_ITER ⇐ ⌈(p − 1)/2⌉
9: while k < MAX_ITER do
10:    k ⇐ k + 1;
11:    if k = 1 then
12:       MPI_SENDRECV(Y, sendTo, Y_r, recvFrom);
13:    else
14:       Y_s ⇐ Y_r;
15:       MPI_SENDRECV(Y_s, sendTo, Y_r, recvFrom);
16:    end if
17:    Do Computation();
18: end while
```

In this chapter we have used observations which consist of randomly selected up to 4 million chemical subsets for our testing. The computing cluster systems we have used in our experiments are summarized in Table 4.1.

In the following, we will mainly show: i) exploration of the optimal number of nearest neighbors; ii) the quality of the proposed MI-MDS interpolation approach in performing MDS algorithms, with respect to various sample sizes – 12.5k, 25k, and 50k randomly selected from 100k dataset as a basis – as well as the mapping results of large-scale data, i.e. up to 4 million points; and iii) parallel performance measurements of our parallelized interpolation algorithms on our clustering systems as listed in Table 4.1; and finally, iv) our results on processing up to 4 million MDS maps based on the trained result from 100K dataset.

4.4.1 Exploration of optimal number of nearest neighbors

Generally, the quality of *k*-NN (*k*-nearest neighbor) classification (or regression) is related to the number of neighbors. For instance, if we choose a larger number for the *k*, then the algorithm shows a higher bias but lower variance. On the other hands, the *k*-NN algorithm shows a lower bias but a higher variance based on a

Table 4.1: 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

smaller number of neighbors. For the case of *k*-NN classification, the optimal number of nearest neighbors (*k*) can be determined by the *N-fold cross validation method* [42] or *leave-one-out cross validation method*, and usually the value that minimizes the cross validation error is picked.

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

Figure 4.2 shows the comparison of mapping quality between the MI-MDS results of 100K data with 50k sample data size in terms of different *k* values. The y-axis of the plot is the *normalized STRESS* value which is divided by $\sum_{i<j} \delta_{ij}^2$. The normalized STRESS value is equal to ONE when all the mapping is at the same position, in that the normalized STRESS value denotes the relative portion of the squared distance error rates of the given data set without regard to various scales of δ_{ij} due to data difference. The equation of normalized STRESS is shown in Eq. (4.17) below.

$$\sigma(X) = \sum_{i<j\leq N} \frac{1}{\sum_{i<j} \delta_{ij}^2} (d_{ij}(X) - \delta_{ij})^2 \tag{4.17}$$

Figure 4.2 shows an interesting result that the optimal number of nearest neighbors is ‘two’ rather than

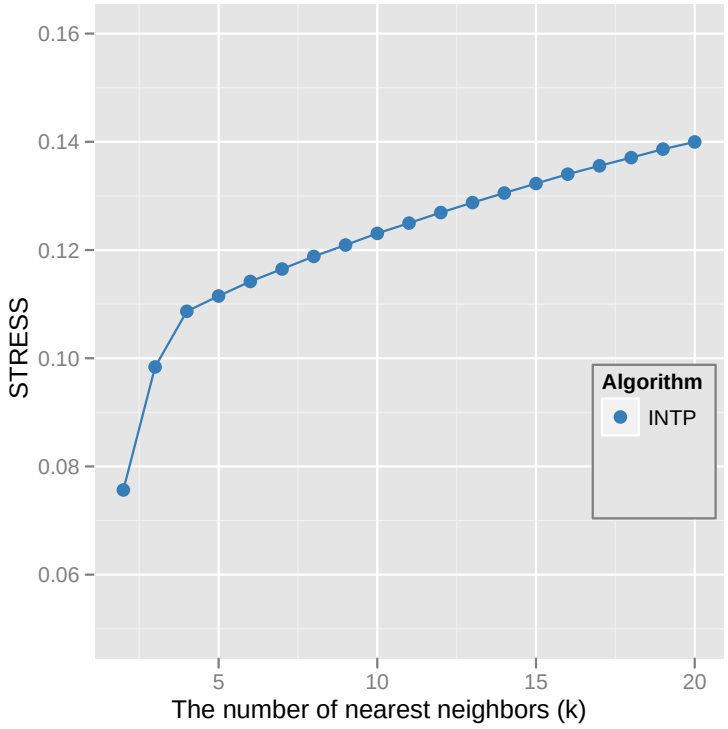
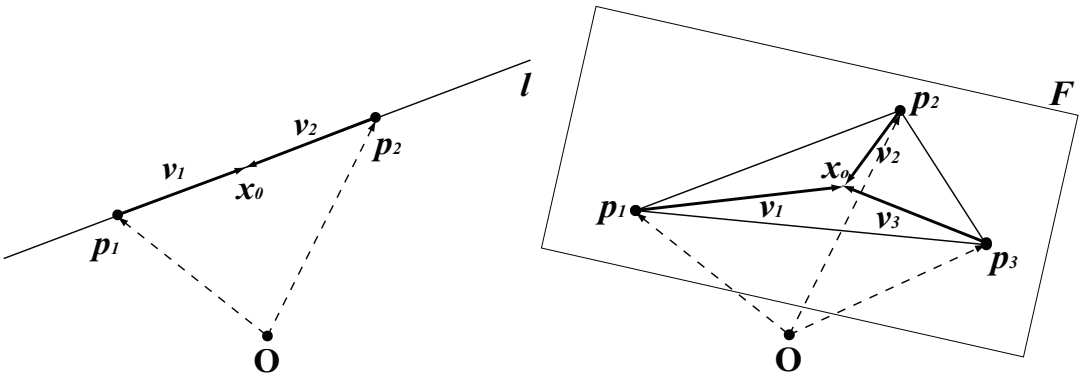


Figure 4.2: Quality comparison between interpolated result of 100k with respect to the number of nearest neighbors (k) with 50k sample and 50k out-of-sample result.

larger values. Also, the normalized STRESS value is statically increased as k is increased, when $k = 5$ and larger, and the normalized STRESS value of MI-MDS results with 20-NN is almost double of that with 2-NN.

Before we analyze the optimal number of nearest neighbors for the proposed MI-MDS method with the given Pubchem dataset, I would like to mention how the proposed MI-MDS solves the mapping ambiguity problem when $k = 2, 3$ for three dimensional target space. When the target dimension is 3D space, logically, the optimal position of the interpolated points can be in a circle if $k = 2$, and the optimal position of the interpolated points can be at two symmetric positions with respect to the face contains all three nearest neighbors, in the case of $k = 3$. The derivative MI-MDS equation in Eq. (4.15), however, constrains the interpolation space corresponding to the number of nearest neighbors, by setting the initial position as the average of the mappings of nearest neighbors. In the case of $k = 2$, the interpolation space is constructed



(a) k = 2 (b) k = 3

Figure 4.3: The illustration of the constrained interpolation space when $k = 2$ or $k = 3$ by initialization at the center of the mappings of the nearest neighbors.

as a line (l) which includes the mapping positions of the two nearest neighbors, when the initial mapping of the interpolation is the center of nearest neighbors ($\overline{p}$). Similarly, the possible mapping position of the interpolated point is constrained within the Face (F) when it contains the three nearest neighbors when $k = 3$.

Figure 4.3-(a) and (b) illustrate the constrained interpolation space in case of $k = 2$ and 3, correspondingly. In Figure 4.3, x_0 represents the initial mapping position of the interpolated point which is the same as $\overline{p}$ and v_i ($i = 1,2$ or 3) is the vector representation of $x_c - p_i$, where x_c is the current mapping position of the interpolated point and p_i is the mapping position in target dimension of nearest neighbors. Note that x_0 ($= \overline{p}$) is on the line l when $k = 2$ and on the face F when $k = 3$. If v_1 and v_2 are on the same line l, $\alpha v_1 + \beta v_2$ is also on the same line l. Similarly, if v_1, v_2, and v_3 are on the same Face F, $\alpha v_1 + \beta v_2 + \gamma v_3$ is also on the same face F. Thus, the final mapping position of the interpolated point with $k = 2$ or 3 is constrained in the line l or face F, as shown in Figure 4.3. This results in removing the ambiguity of the optimal mapping position of the small nearest neighbor cases, for example $k = 2,3$ when the target dimension is 3.

We can think of two MI-MDS specific properties as possible reasons for the results of the experiment of the optimal number of nearest neighbors which is shown in Figure 4.2. A distinct feature of MI-MDS algorithm compared to other k-NN approaches is that the increase of the number of nearest neighbors results

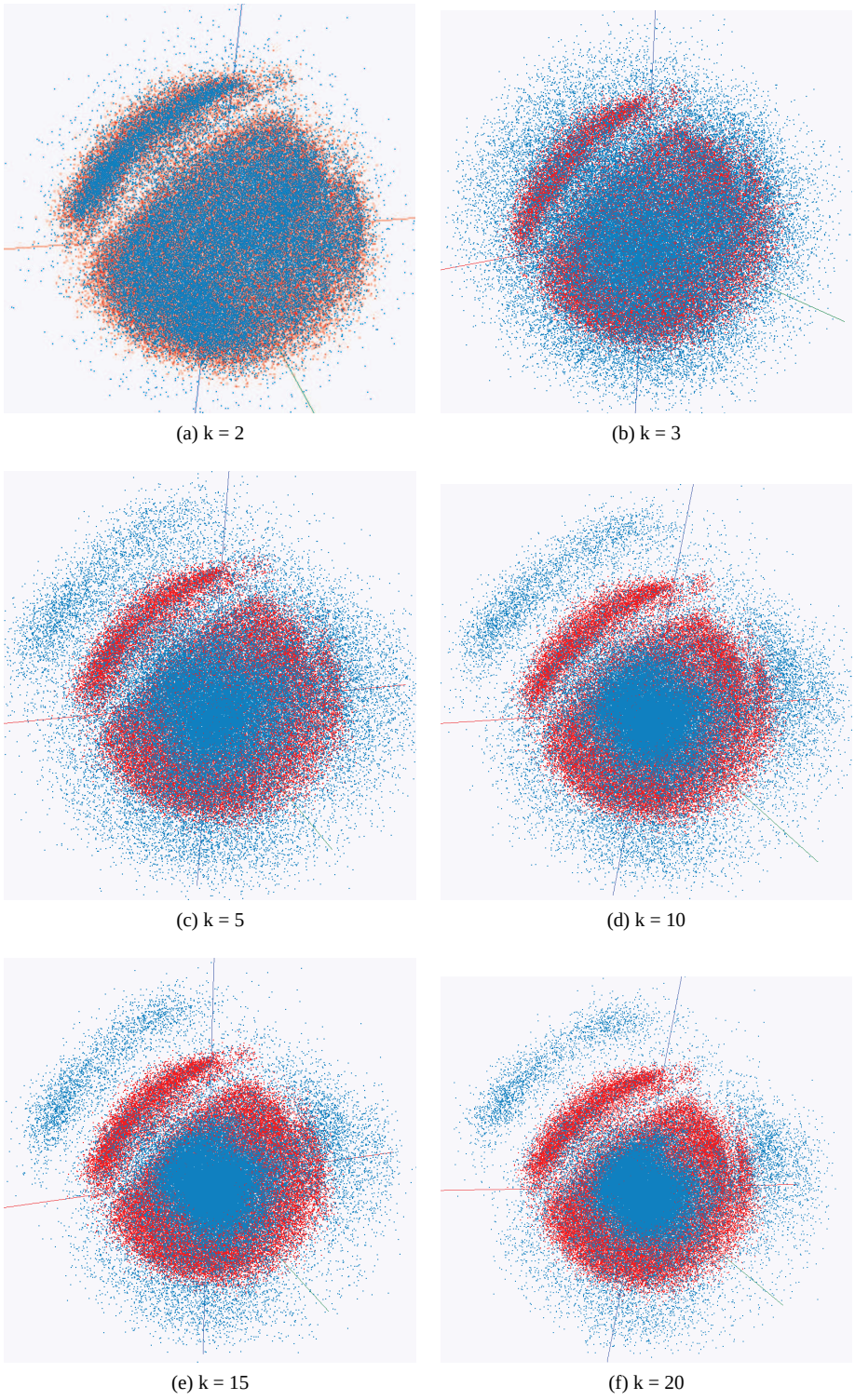
Table 4.2: Analysis of Maximum Mapping Distance between *k*-NNs with respect to the number of nearest neighbors (*k*).

#k-NNs	$<$ **3.0**	$>$ **6.0**	$>$ **7.0**	$>$ **8.0**	**% of (**$<$ **3.0)**	**% of (**$>$ **6.0)**
2	45890	409	164	53	91.780	0.818
3	41772	916	387	139	83.544	1.832
5	34503	1945	867	334	69.006	3.890
10	22004	4230	2005	826	44.008	8.460
20	10304	8134	4124	1797	20.608	16.268

in generating more a complicated problem space to find the mapping position of the newly interpolated point. Note that the interpolation approach allows only the interpolated point to be moved and the selected nearest neighbors are fixed in the target dimension. This algorithmic property effects more severe constraints to find optimal mapping position with respect to Eq. (4.2). Also, note that finding the optimal interpolated position does not guarantee that it makes better mapping in terms of full data mapping, but it means that MI-MDS algorithm works as the algorithm designed.

Another specific property of MI-MDS is that the purpose of the MI-MDS algorithm is to find appropriate embeddings for the new points based on the given mappings of the sample data. Thus, it could be better to be sensitive to the mappings of closely-located nearest neighbors of the new point than to be biased to the distribution of the mappings of whole sample points. Figure 4.4 illustrates the mapping difference with respect to the number of nearest neighbors used for MI-MDS algorithm with 50k sample and 50k out-of-sample data. The 50k sample data is selected randomly from the given 100k data set, so it is reasonable that the sampled 50k data and out-of-sample 50k show similar distributions. As shown in Figure 4.4-(a), the interpolated points are distributed similar to the sampled data as we expected. Also, Figure 4.4-(a) are much more similar to the configuration of the full MDS running with 100k data, which is shown later in this chapter, than other results in Figure 4.4. On the other hand, Figure 4.4-(b) through Figure 4.4-(f) are shown in center-biased mapping, and the degree of bias of those mappings increases as *k* increases.

In order to understand more about why biased mappings are generated by larger nearest neighbors cases



(a) k = 2 (b) k = 3

(c) k = 5 (d) k = 10

(e) k = 15 (f) k = 20

Figure 4.4: The mapping results of MI-MDS of 100k Pubchem data with 50k sample data and 50k out-of-sample data with respect to the number of nearest neighbors (k). The sample points are shown in red and the interpolated points are shown in blue.

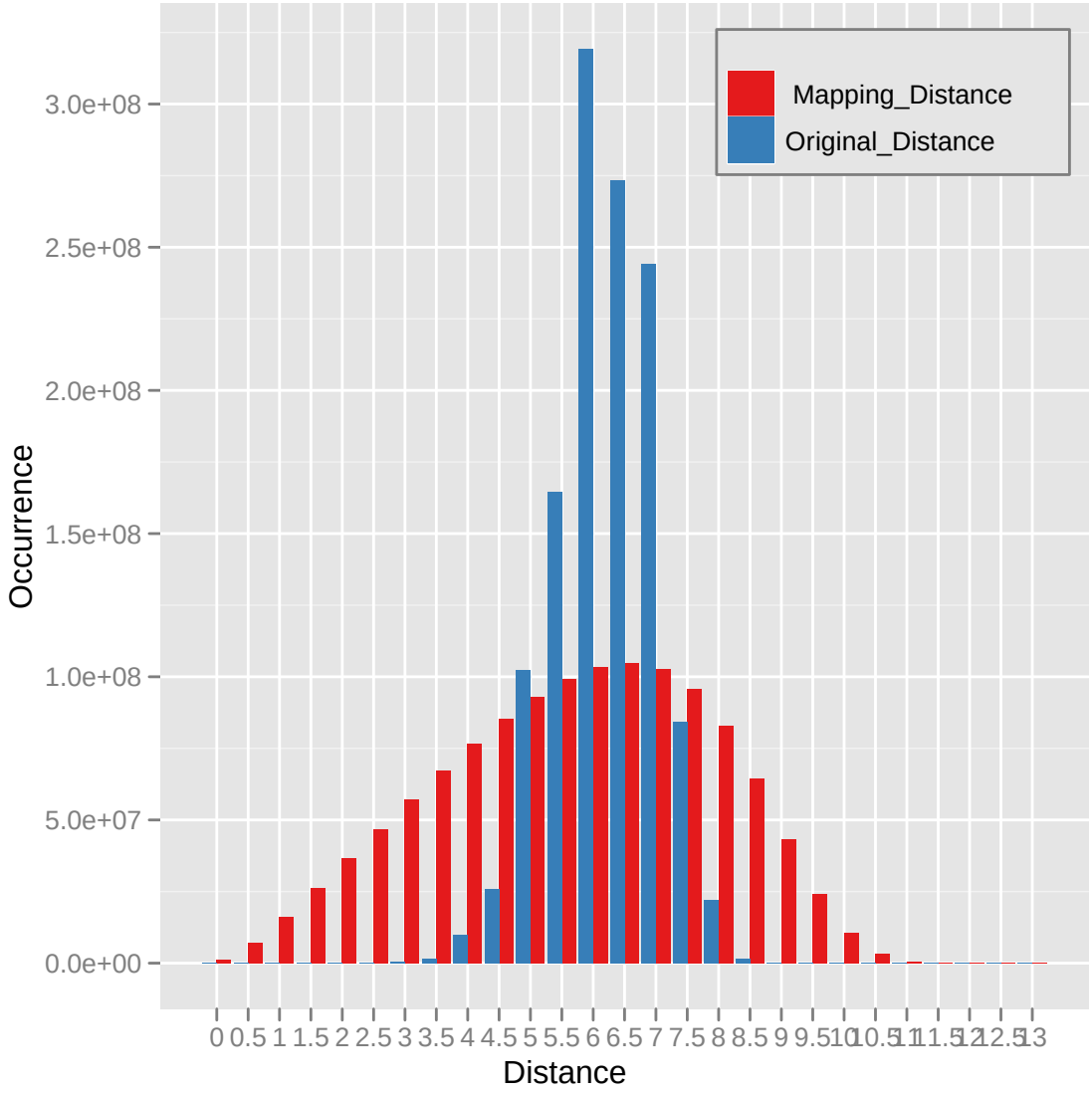


Figure 4.5: Histogram of the original distance and the pre-mapping distance in the target dimension of 50k sampled data of 100k. The maximum original distance of the 50k sampled data is 10.198 and the maximum mapping distance of the 50k sampled data is 12.960.

with the test dataset, we have investigated the given original distance distribution of the 50k sampled data set and the trained mapping distance distribution of the sampled data. Also, we have analyzed the training mapping distance between k-NNs with respect to k. Figure 4.5 is the histogram of the original distance distribution and the trained mapping distance distribution of 50k sampled data used in Figure 4.2 and Figure 4.4. As shown in Figure 4.5, most of the original distances are in between 5 and 7, but the trained mapping distances reside in a more broad interval. Table 4.2 demonstrates the distribution of the maximum mapping distance between selected k-NNs with respect to the number of nearest neighbors. The maximum original distance is 10.198 and the maximum mapping distance of the 50k sampled data is 12.960.

As shown in Figure 4.16-(a), the mapping of Pubchem data forms a spherical shape. Thus, the maximum mapping distance of the 50k sampled data could be similar to the diameter of the spherical mapping. The distance 3.0 is close to the half of radius of the sphere and the distance 6.0 is close to the radius of the sphere. Therefore, in Table 4.2, the column of "$<$ **3.0**" represents the cases that nearest neighbors are closely mapped together, and the columns of "$>$ **6.0**" and others illustrate the cases that some nearest neighbors are far from other nearest neighbors. Note that the entries of "$>$ **6.0**" column include that of "$>$ **7.0**" and "$>$ **8.0**" as well.

The analysis of mapping distance between k-NNs with the tested Pubchem dataset shows interesting results. Initially, we expected that $k = 5$ or $k = 10$ could be small enough numbers of the nearest neighbors, which would make nearest neighbors be positioned near each other in the training mapping results. Contrary to our expectation, as shown in Table 4.2, even in the case of $k = 2$, nearest neighbors are not near each other for some interpolated data. The cases of two nearest neighbors positioned more than a 6.0 distance occurred more than 400 times. As we increase k to be equal to 3, the occurrence of the cases of at least two nearest neighbors distanced more than 6.0 increases more than twice of what it was when $k = 2$. On the other hand, the number of the cases of all of the selected nearest neighbors closely mapped is decreased in Table 4.2. The percentage of the cases of all of the selected neareset neighbors closely mapped is also shown in Table 4.2. Between the cases of $k = 2$ and $k = 3$, the difference of the all k-NNs closely mapped cases is about 8.2% of

a 50k out-of-sample points. For the case of $k = 20$, the occurrence of closely mapped cases is dropped down from 91.8% to 20.6%.

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

In short, as we explored the optimal number of nearest neighbors with Pubchem data set, $k = 2$ is the optimal case as shown in Figure 4.2 with the Pubchem dataset, and the larger nearest neighbor cases show biased mapping results, as shown in Figure 4.4. Therefore, we use 2-NN for the forthcoming MI-MDS experiments analyzed in this section.

4.4.2 Comparison between MDS and MI-MDS

4.4.2.1 Fixed Full Data Case Figure 4.6 shows the comparison of quality between MI-MDS results of 100K data with different sample data sizes by using 2-NN and MDS (SMACOF) as the only results with the 100k Pubchem data. The y-axis of the plot is the normalized STRESS value which is shown in Eq. (4.17). The normalized STRESS difference between the MDS only results and interpolated with 50k is only around 0.0038. Even with a small portion of sample data (12.5k data is only $1/8$ of 100k), the proposed MI-MDS algorithm produces good enough mapping in the target dimension using a much smaller amount of time than when we ran MDS with a full 100k of data. In Figure 4.7, we compare the commulated running time of the *out-of-sample* approach, which combines the full MDS running time of sample data and MI-MDS running time of the out-of-sample data with respect to different sample size, to the running time of the full MDS run with the 100k data. As shown in Figure 4.7, the overall running time of the out-of-sample approach is much

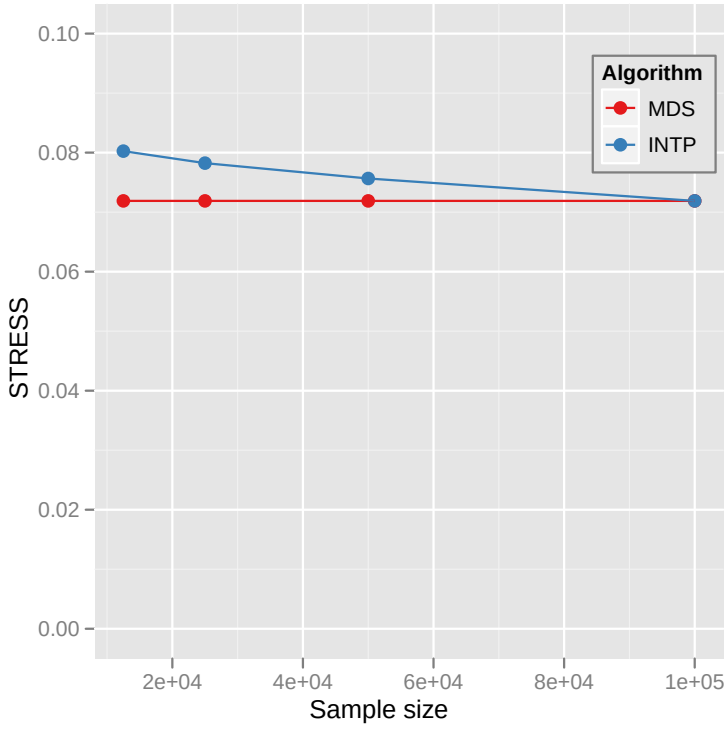


Figure 4.6: Quality comparison between the interpolated result of 100k with respect to the different sample sizes (INTP) and the 100k MDS result (MDS)

smaller than the full MDS approach. To be more specific, the out-of-sample approach for 100k dataset takes around 25.1, 9.3, and 3.3 times faster than the full MDS approach with respect to different sample sizes, 12.5k, 25k, and 50k, correspondingly.

Figure 4.8 shows the MI-MDS interpolation running time only with respect to the sample data using 16 nodes of the Cluster-II in Table 4.1. The MI-MDS algorithm takes around 8.55, 14.35, and 18.98 seconds with different sample sizes, i.e. 12.5k, 25k, and 50k, to find new mappings of 87500, 75000, and 50000 points based on the pre-mapping results of the corresponding sample data. Note that the full MDS running time with 100k using 16 nodes of the Cluster-II in Table 4.1 is around 27006 sec. In Figure 4.8, we can find the interesting feature that it takes much less time to find new mappings of 87,500 points (8.55 seconds) than to find new mappings of 50,000 points (18.98 seconds). The reason is the computational time complexity

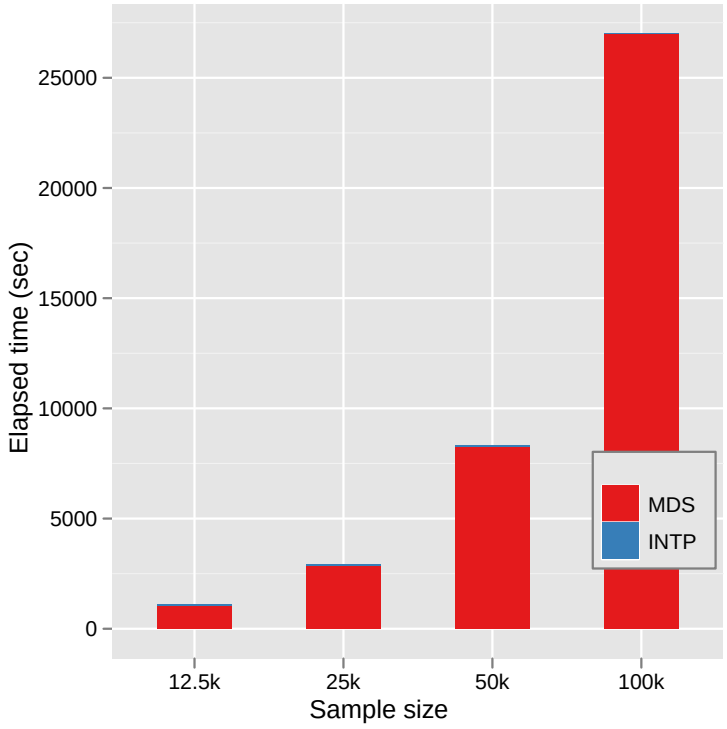


Figure 4.7: Running time comparison between the Out-of-Sample approach which combines the full MDS running time with sample data and the MI-MDS running time with out-of-sample data when $N = 100k$, with respect to the different sample sizes and the full MDS result of the 100k data.

of MI-MDS is $\mathcal{O}(Mn)$ where n is the sample size and $M = N - n$. Thus, the running time of MI-MDS is proportional to the number of new mapping points if the sample size (n) is the same, as in the larger data set case shown below in this chapter. However, the above case is the opposite case. The full data size (N) is fixed, so that both the sample data size (n) and the out-of-sample data size (M) are variable and correlated. We can illustrate $\mathcal{O}(Mn)$ as a simple quadratic equation of variable n as following: $\mathcal{O}(n*(N-n)) = \mathcal{O}(N*n-n^2)$, which has maximum when $n = N/2$. The above experiment case $N = 100k$ and $n = 50k$ is the maximum case, so that the case of 50k sample data of MI-MDS took longer than the case of the 12.5k sample data.

4.4.2.2 Fixed Sample Data Size Above we discussed the MI-MDS quality of the fixed total number (100k) with respect to the different sample data sizes, compared to the MDS running results with total number of

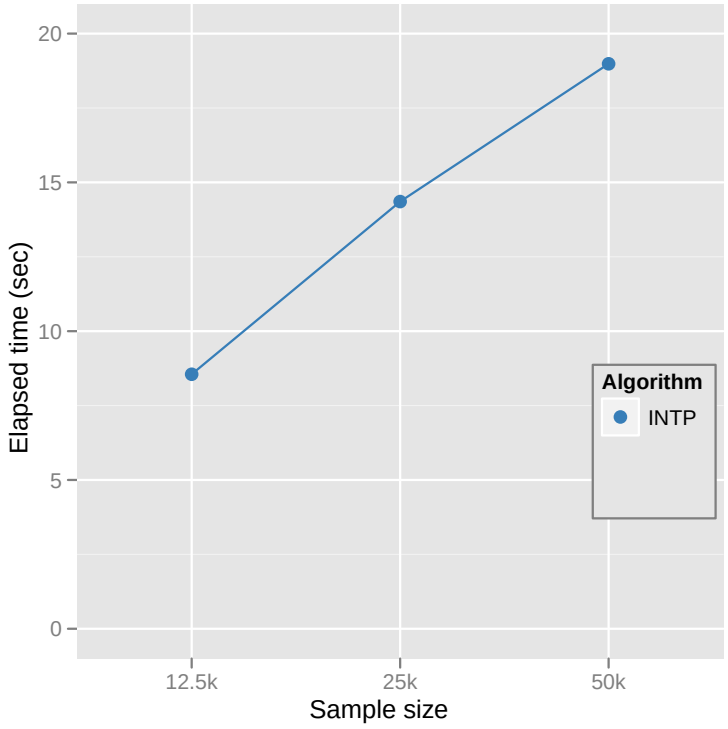


Figure 4.8: Elapsed time of parallel MI-MDS running time of 100k data with respect to the sample size using 16 nodes of the Cluster-II in Table 4.1. Note that the computational time complexity of MI-MDS is $\mathcal{O}(Mn)$ where n is the sample size and $M = N - n$.

data (100k). Now, the opposite direction of the test, which tests the scalability of the proposed interpolation algorithm, is performed as follows: we fix the sample data size to 100k, and the interpolated data size is increased from one million (1M) to two million (2M) to four million (4M). Then, the STRESS value is measured for each running result of total data, i.e. 1M + 100k, 2M + 100k, and 4M + 100k. The measured STRESS value is shown in Figure 4.9. There is some quality lost between the full MDS running results with 100k data and the 1M interpolated results based on that 100k mapping; they have about a 0.007 difference in the normalized STRESS criteria. However, there is not much difference between the normalized STRESS values of the 1M, 2M, and 4M interpolated results, although the sample size is quite a small portion of the total data and the out-of-sample data size increases as quadruple the numbers. From the above results, we could consider that the proposed MI-MDS algorithm works well and is scalable if we are given a good enough

Table 4.3: Large-scale MI-MDS running time (seconds) with 100k sample data

1 Million	2 Million	4 Million
731.1567	1449.1683	2895.3414

pre-configured result which represents well the structure of the given data. Note that it is not possible to run the SMACOF algorithm with only 200k data points due to memory bounds, within the systems in Table 4.1.

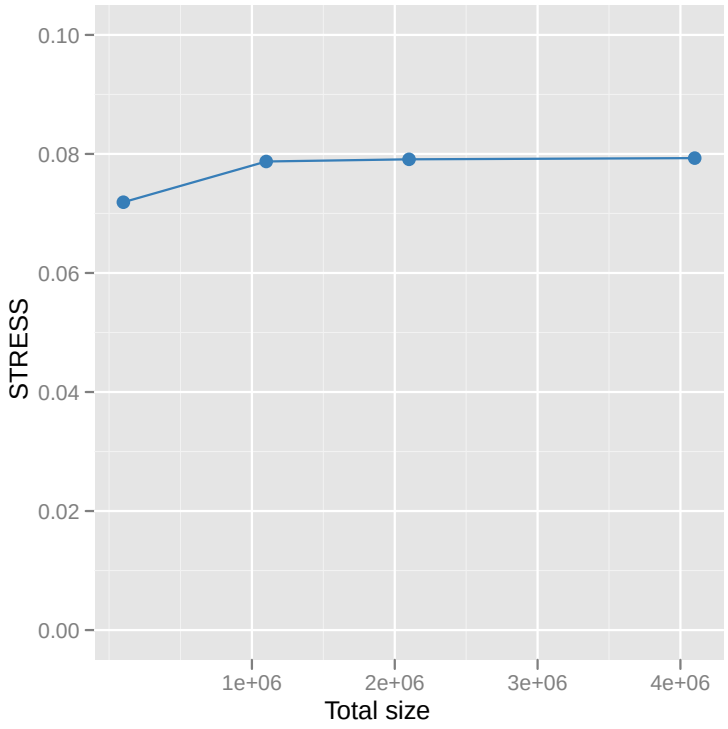


Figure 4.9: The STRESS value change of the interpolation larger data, such as 1M, 2M, and 4M data points, with 100k sample data. The initial STRESS value of MDS result of 100k data is 0.0719.

We also measure the runtime of the MI-MDS algorithm with a large-scale data set up to 4 million points. Figure 4.10 shows the running time of the out-of-sample approach in a commulated bar graph, which represents the full MDS running time of sample data ($M = 100$k) in the red bar and the MI-MDS interpolation time of out-of-sample data ($n =$ 1M, 2M, and 4M) in the blue bar on top of the red bar. As we expected, the running time of MI-MDS is much faster than the full MDS running time in Figure 4.10. Although the

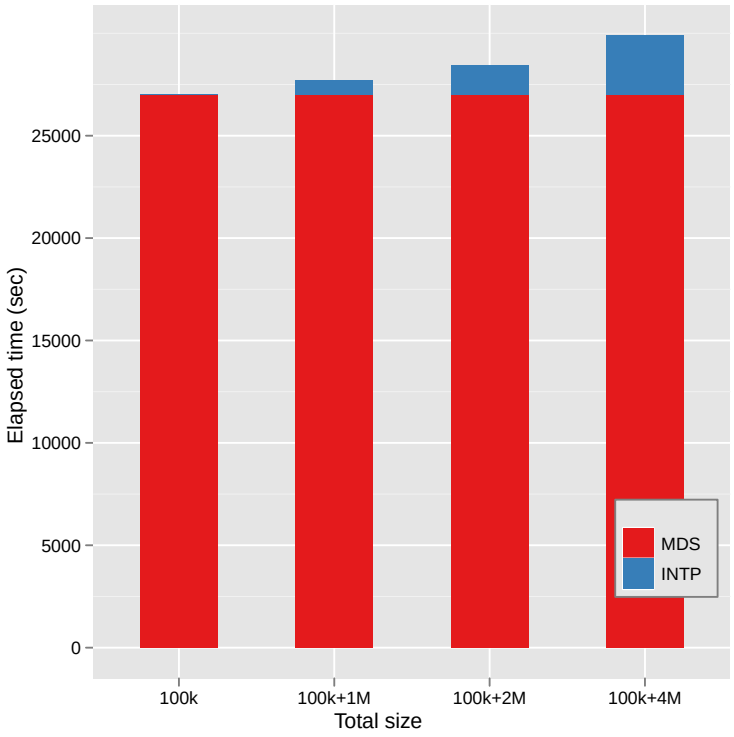


Figure 4.10: Running time of the Out-of-Sample approach which combines the full MDS running time with sample data ($M = 100k$) and the MI-MDS running time with different out-of-sample data sizes, i.e. 1M, 2M, and 4M.

MI-MDS interpolation running time in Table 4.3 is much smaller than the full MDS running time (27006 seconds), the MI-MDS deals with a much larger amount of points, i.e. 10, 20, and 40 times larger number of points. Note that we cannot run the parallel SMACOF algorithm [16] with even 200,000 points on our current sytsems in Table 4.1. Even though we assume that we are able to run the parallel SMACOF algorithm with millions of points on **Cluster-II** in Table 4.1, the parallel SMACOF will take 100, 400, and 1600 times longer with 1M, 2M, and 4M data than the running time of parallel SMACOF with 100k data, due to the $\mathcal{O}(N^2)$ computational complexity. As opposed to the approximated full MDS running time, the proposed MI-MDS interpolation takes much less time to deal with millions of points than parallel SMACOF algorithm. In numeric, MI-MDS interpolation is faster than approximated full parallel MDS running time in 3693.5, 7454.2, and 14923.8 times with 1M, 2M, and 4M data, correspondingly.

If we extract the MI-MDS running time only with respect to the out-of-sample data size from Figure 4.10, the running time should be proportional to the number of out-of-sample data since the sample data size is fixed. Table 4.3 shows the exact running time of the MI-MDS interpolation method with respect to the number of the out-of-sample data size (n), based on the same sample data ($M = 100$k). The running time is almost exactly proportional to the out-of-sample data size (n), as it should be.

4.4.3 Parallel Performance Analysis of MI-MDS

In the above section, we discussed the quality of the constructed configuration of the MI-MDS approach based on the STRESS value of the interpolated configuration, and the running time benefits of the proposed MI-MDS interpolation approach. Here, we would like to investigate the MPI communication overhead and parallel performance of the proposed parallel MI-MDS implementation in Section 4.3.1 in terms of efficiency with respect to the running results within Cluster-I and Cluster-II in Table 4.1.

First of all, we prefer to investigate the parallel overhead, especially the MPI communication overhead, which could be major parallel overhead for the parallel MI-MDS in Section 4.3.1. Parallel MI-MDS consists of two different computations, the MI part and the STRESS calculation part. The 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. 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 results to all the processes for 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 were applied to the proposed symmetric pairwise computation in Section 4.3.2, each process uses `MPI_SENDRECV` k times to send an assigned block or rolled block, whose size is M/p, where $k = \lceil (p-1)/2 \rceil$ for communicating required data and `MPI_REDUCE` twice for

calculating $\sum_{i<j}(d_{ij}-\delta_{ij})^2$ and $\sum_{i<j}\delta_{ij}^2$. Thus, the MPI communicated data size is $\mathcal{O}(M/p \times p) = \mathcal{O}(M)$ without regard to the number of processes.

The MPI overhead during the MI part and the STRESS calculating part at Cluster-I and Cluster-II in Table 4.1 are shown in Figure 4.11 and Figure 4.12, correspondingly. Note that the x-axis of both figures is the sample size (n) but not $M = N - n$. In the figures, the model is generated as $\mathcal{O}(M)$ starting with the smallest sample size, here 12.5k. Both Figure 4.11 and Figure 4.12 show that the actual overhead measurement follows the MPI communication overhead model.

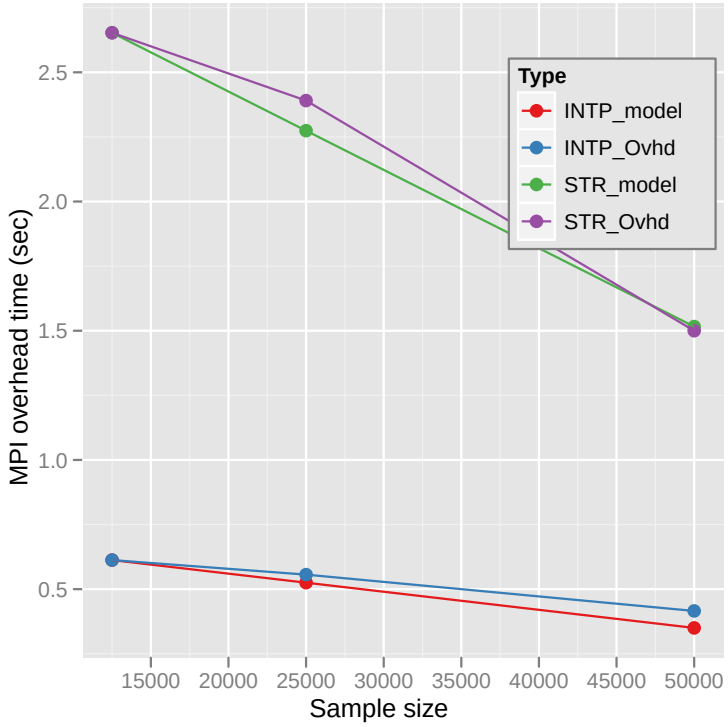


Figure 4.11: Parallel overhead modeled as due to MPI communication in terms of sample data size (m) using Cluster-I in Table 4.1 and message passing overhead model.

Figure 4.13 and Figure 4.14 illustrate the efficiency of the interpolation part and the STRESS calculation part of the parallel MI-MDS running results with different sample size - 12.5k, 25k, and 50k - with respect to the number of parallel units using Cluster-I and Cluster-II, correspondingly. Equations for the efficiency is

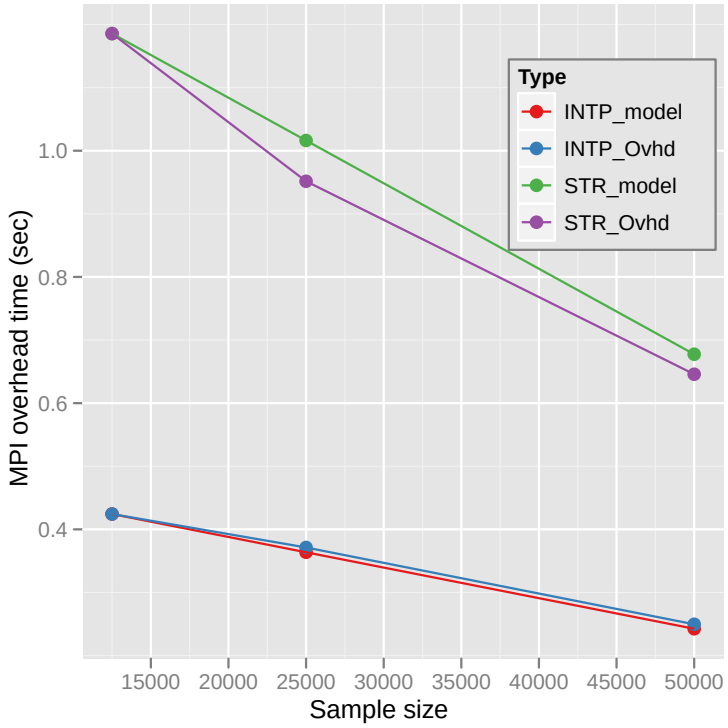


Figure 4.12: Parallel overhead modeled as due to MPI communication in terms of sample data size (m) using Cluster-II in Table 4.1 and message passing overhead model.

follows:

$$f = \frac{pT(p) - T(1)}{T(1)} \tag{4.18}$$

$$\varepsilon = \frac{1}{1+f} \tag{4.19}$$

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. (4.18) can be replaced with following:

$$f = \frac{\alpha T(p_1) - T(p_2)}{T(p_2)} \tag{4.20}$$

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. (4.20) for the overhead calculation.

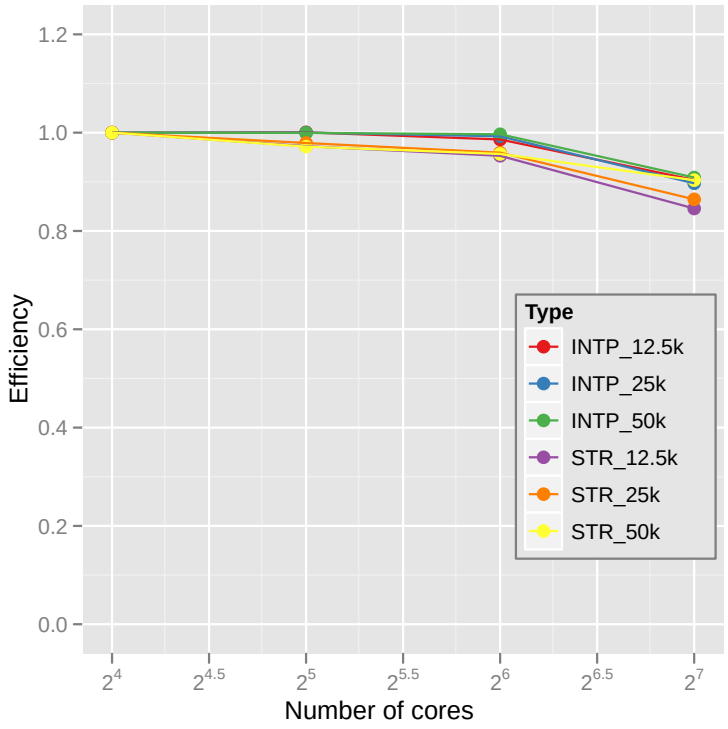


Figure 4.13: Efficiency of the interpolation part (INTP) and the STRESS evaluation part (STR) runtimes in the parallel MI-MDS application with respect to different sample data sizes using Cluster-I in Table 4.1. The total data size is 100K.

In Figure 4.13, 16 to 128 cores are used to measure parallel performance with 8 processes, and 32 to 384 cores are used to evaluate the parallel performance of the proposed parallel MI-MDS with 16 processes in Figure 4.14. Processes communicate via MPI primitives and each process is also parallelized at the thread level. Both Figure 4.13 and Figure 4.14 show very good efficiency with an appropriate degree of parallelism. Since both the MI part and the 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 the 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

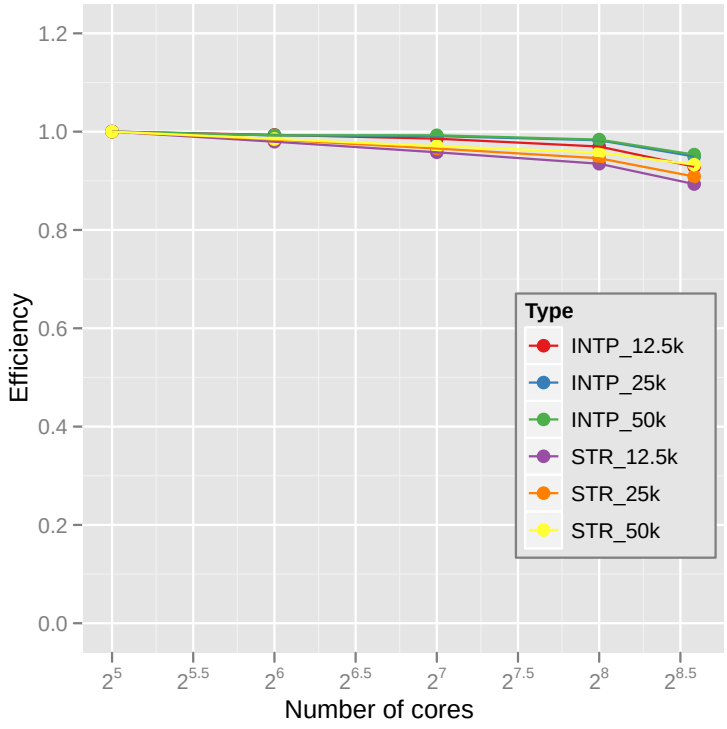
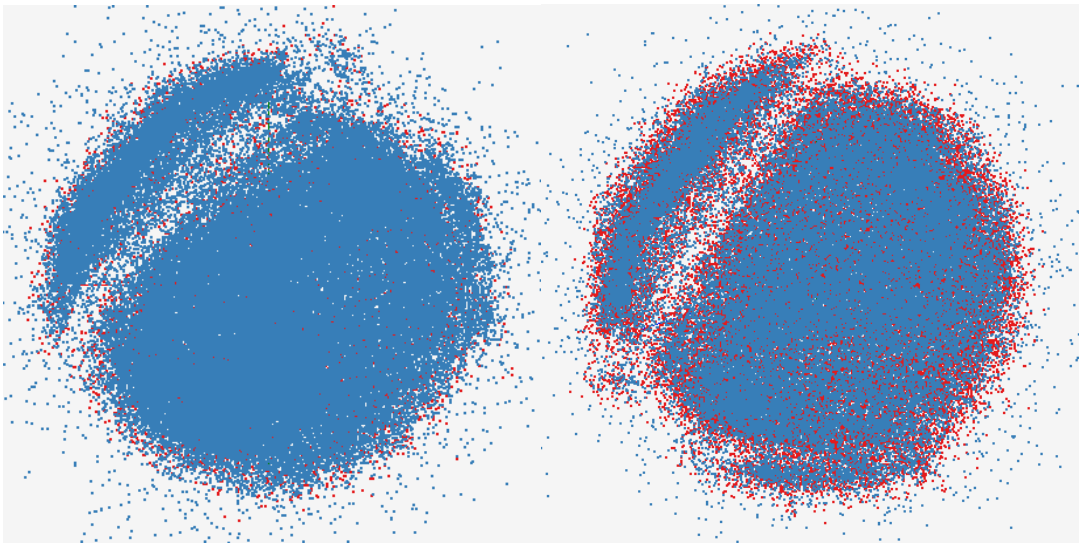


Figure 4.14: Efficiency of the interpolation part (INTP) and the STRESS evaluation part (STR) runtimes in the parallel MI-MDS application with respect to different sample data sizes using Cluster-II in Table 4.1. The total data size is 100K.

constant if we examine it with the same number of processes and the same out-of-sample data sizes. Since the parallel computation time decreases as more cores are used, but the overhead time remains constant, this dynamic lowers the efficiency as the number of cores is increased, as we expected. Note that the number of processes which lowers the efficiency dramatically is different between the Cluster-I and Cluster-II. The reason is that the MPI overhead time of Cluster-I is bigger than that of Cluster-II due to different network environments, i.e. Giga bit ethernet and 20Gbps Infiniband. The difference is easily found by comparing Figure 4.11 and Figure 4.12.

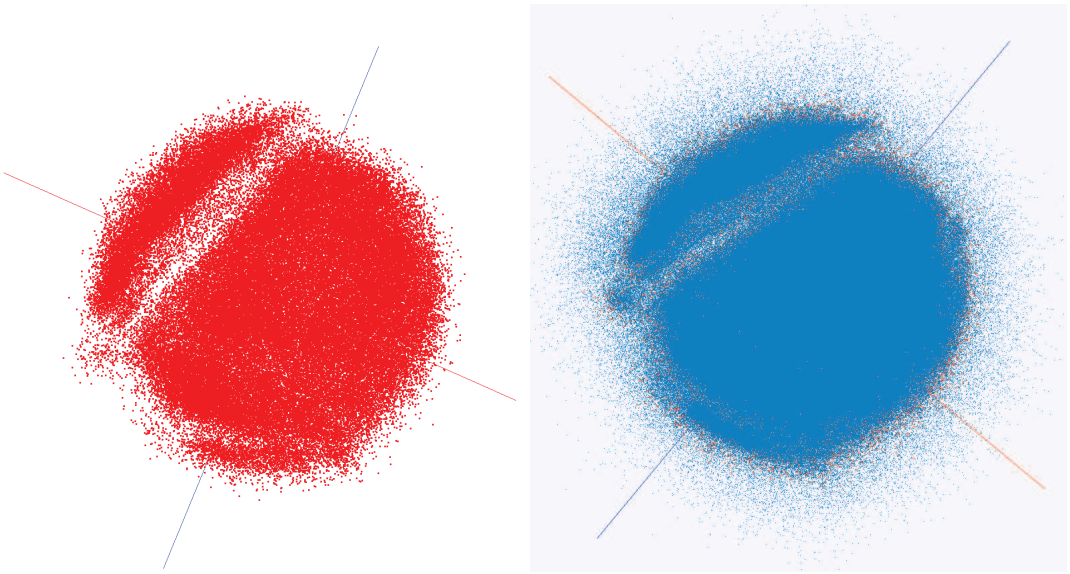


(a) MDS 12.5k (b) MDS 50k

Figure 4.15: Interpolated MDS results of total 100k PubChem dataset trained by (a) 12.5k and (b) 50k sampled data. Sampled data are colored in red and interpolated points are in blue.

4.4.4 Large-Scale Data Visualization via MI-MDS

Figure 4.15 shows the proposed MI-MDS results of a 100k PubChem dataset with respect to the different sample sizes, such as (a) 12.5k and (b) 50k. Sampled data and interpolated points are colored in red and blue, correspondingly. With our parallel interpolation algorithms for MDS, we have also processed a large volume of PubChem data by using our Cluster-II, and the results are shown in Figure 4.16. We performed parallel MI-MDS to process datasets of hundreds of thousand and up to 4 million by using the 100k PubChem data set as a training set. In Figure 4.16, we show the MI-MDS result of 2 million dataset based on 100k training set, compared to the mapping of 100k training set data. The interpolated points are colored in blue, while the training points are in red. As one can see, our interpolation algorithms have produced a map closed to the training dataset.



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

Figure 4.16: Interpolated MDS results. Based on 100k samples (a), additional 2M PubChem dataset is interpolated (b). Sampled data are colored in red and interpolated points are in blue.

4.5 Summary

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

Before starting a comparative experimental analysis between MI-MDS and the full MDS algorithm, we explored the optimal number of k-NN. 2-NN is the best case for the Pubchem data which we used in this

chapter. We have shown that our interpolation approach gives results of good quality with high parallel performance. In a quality comparison, the experimental results shows that the interpolation approach output is comparable to the normal MDS output, which takes much more running time than interpolation. The proposed interpolation algorithm is easy to parallelize since each interpolated points is independent of other out-of-sample points, so many points can be interpolated concurrently without communication. The parallel performance is analyzed in Section 4.4.3, and it shows very high efficiency as we expected.

Consequently, the interpolation approach enables us to configure 4 millions Pubchem data points in this chapter with an acceptable normalized STRESS value, compared to the normalized STRESS value of 100k sampled data in less than ONE hour, and the size can be extended further with a moderate running time. Note that if we use parallel MDS only, we cannot even run with only 200,000 points on the Cluster-II system in Table 4.1 due to the out of memory exception, and if it were possible to run parallel MDS with 4 million data points on the Cluster-II system, it would take around 15,000 times longer than the interpolation approach as mentioned in Section 4.4.2. Future research includes application of these ideas to different areas including metagenomics and other DNA sequence visualization.

5

Deterministic Annealing SMACOF

5.1 Overview

Multidimensional scaling (MDS) [13,45] is a well-known dimension reduction algorithm which is a non-linear optimization problem constructing a lower dimensional configuration of high dimensional data with respect to the given pairwise proximity information based on an objective function, namely STRESS [44] or SSTRESS [67]. Below equations are the definition of STRESS (5.1) and SSTRESS (5.2):

$$\sigma(X) = \sum_{i<j\leq N} w_{ij}(d_{ij}(X) - \delta_{ij})^2 \tag{5.1}$$

$$\sigma^2(X) = \sum_{i<j\leq N} w_{ij}[(d_{ij}(X))^2 - (\delta_{ij})^2]^2 \tag{5.2}$$

where $1 \leq i < j \leq N$, w_{ij} is a weight value ($w_{ij} \geq 0$), $d_{ij}(X)$ is a Euclidean distance between mapping results of x_i and x_j, and δ_{ij} is the given original pairwise dissimilarity value between x_i and x_j. SSTRESS is adopted by ALSCAL algorithm (Alternating Least Squares Scaling) [67], and uses *squared* Euclidean distances results in simple computations. A more natural choice could be STRESS which is used by SMACOF [20] and Sammon's mapping [64].

Due to the non-linear optimization properties of the MDS problem, many heuristic optimization methods have been applied to solve the MDS problem. An optimization method called iterative majorization is used to solve the MDS problem by a well-known MDS algorithm called SMACOF [20,21]. The iterative majorization method is a type of Expectation-Maximization (EM) approach [22], and it is well understood that EM method suffers from local minima problem although EM method is widely applied to many optimization problems.

From the motivation of local-optima avoidance, I have proposed an MDS algoirthm which applies a robust optimization method called Deterministic Annealing (DA) [62, 63] to the MDS problem, in this chapter. A key feature of the DA algorithm is to endeavour to find global optimum in a *deterministic way* [63] without trapping local optima, instead of using a stochastic random approach, which results in a long running time, as in Simulated Annealing (SA) [40]. DA uses *mean field approximation*, which is calculated in a deterministic way, by using the statistical physics integrals.

In Section 5.2, I briefly discuss various optimization methods which have been applied to the MDS problem to avoid local optima. Then, the proposed DA MDS algorithm based on iterative majorization method is explained in Section 5.3. Section 5.4 and Section 5.5 illustrate performance of the proposed DA MDS algorithm compared to other MDS algorithms with variable data sets followed by the conclusion in Section 5.6.

5.2 Related Work

5.2.1 Avoiding Local Optima in MDS

As I mentioned above, many heuristic optimization methods are applied to solve the MDS problem, which is a non-linear optimization problem. In this section, I would like to summarize those various optimization approaches which have been applied to MDS problem.

SMACOF is a quite useful algorithm, since it monotonically decreases the STRESS criterion [20] by each iteration. However, the well-known problem of the gradient descent approach is to be trapped in a local minima due to its hill-climbing approach, and it is applied to SMACOF as well. In order to avoid the local optima problem, stochastic optimization approaches, such as simulated annealing (SA) [40] and genetic algorithms (GA) [29, 37], have been used for solving the MDS problem [14, 49, 50, 72, 76], but stochastic algorithms are well-known to suffer from a long running time due to their Monte Carlo approach. In addition to stochastic algorithms, the distance smoothing [35] and the tunneling method [34] for MDS problem were proposed to avoid local optima in a deterministic way.

Recently, Ingram et al. introduced a multilevel algorithm called Glimmer [38] which is based on a force-based MDS algorithm with restriction, relaxation, and interpolation operators. Glimmer shows less sensitivity to initial configurations than the GPU-SF subsystem, which is used in Glimmer [38], due to the multilevel nature. In Glimmer's paper [38], however, the SMACOF algorithm shows better mapping quality than Glimmer. Also, the main purpose of Glimmer is to achieve a speeded up running time with less cost of quality degradation rather than it being explicitly focused on improving mapping quality. By contrast, this chapter focuses on an optimization method which improves mapping quality in a deterministic approach. Therefore, we will compare the proposed algorithm to other optimization algorithms, i.e. the SMACOF [20] and the Distance Smoothing method [35], in Section 5.4 and Section 5.5.

In addition to many optimization methods mentioned above, deterministic annealing (DA) [62, 63] is also a well-known optimization method. DA method is used for many optimization problems, including clustering [62, 63], pairwise clustering [36], and MDS [41], to name a few. Since it is intractable to calculate $\mathscr{F}$ in Eq. (2.21) exactly, an approximation technique called *mean field approximation* is used to solve the MDS problem by DA in [41], in that Gibbs distribution $P^G(X)$ is approximated by a factorized distribution with density

$$P^0(X|\Theta) = \prod_{i=1}^{N} q_i(x_i|\Theta_i). \tag{5.3}$$

where Θ_i is a vector of mean field parameter of x_i and $q_i(x_i|\Theta_i)$ is a factor serves as a marginal distribution model of the coordinates of x_i. To optimize parameters Θ_i, Klock and Buhmann [41] minimized Kullback-Leibler (KL) divergence between the approximated density $P^0(X)$ and the Gibbs density $P^G(X)$ through EM algorithm [22].

5.3 Deterministic Annealing SMACOF

Although DA-MDS [41] shows the general approach of applying DA to the MDS problem through mean field approximation, it is not clearly explained how it can solve MDS in the paper [41]. While the authors mentioned that the Gibbs density is approximated by a factorized distribution with density (Eq. (5.3)) in [41], they did not clearly pose which model densities are practically applicable to proceeding the mean field approximation in order to solve the MDS problem in their paper [41]. Therefore, I developed an alternative way to solve the MDS problem by using a DA optimization method, and will introduce the alternative way to utilize the DA method to the MDS problem in this section.

Klock and Buhmann applied deterministic annealing to the MDS problem by defining an approximation of Gibbs distribution ($P^0(X)$) via factorized distribution as shown in Eq. (5.3). Then, they developed the solution for the MDS problem for the given data by minimizing the Kullback-Leibler (KL) divergence between the factorial distribution ($P^0(X)$) and the Gibbs distribution ($P^G(X)$).

On the other hand, the alternative DA method for the MDS problem was initiated by defining a tractable expected energy function of MDS problem, which is based on a simple Gaussian distribution, as shown in Eq. (5.5) below. Then, the Gibbs distribution is approximated by exchanging the newly defined expected energy function ($\mathscr{H}_0$) into $\mathscr{H}(X)$ of Eq. (2.20) as shown in Eq. (5.6). Finally, the proposed algorithm finds

an MDS solution for a given data set by minimizing $\mathscr{F}_{MDS}(P^0)$.

Based on the experimental results with various data sets, shown in Section 5.4 and Section 5.5, the proposed DA MDS algorithm can be considered as the best algorithm which finds better quality mapping in a deterministic way than other deterministic MDS solutions with respect to the quality and the consistency of the outputs. Also, it even shows some efficiency with larger experimental data sets compared to the EM-like algorithm.

From now on, I would like to introduce the details of the proposed DA approach for the MDS problem in this chapter. If we use the STRESS (5.1) objective function as an expected energy (cost) function in Eq. (2.22), then we can define $\mathscr{H}_{MDS}$ and $\mathscr{H}_0$ as following:

$$\mathscr{H}_{MDS} = \sum_{i<j\leq N}^{N} w_{ij}(d_{ij}(X)-\delta_{ij})^2 \tag{5.4}$$

$$\mathscr{H}_0 = \sum_{i=1}^{N} \frac{(x_i-\mu_i)^2}{2} \tag{5.5}$$

where $\mathscr{H}_0$ corresponds to an energy function based on a simple multivariate Gaussian distribution and μ_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 $\mathscr{F}_0$ as following:

$$P^0(X) = \exp\left(-\frac{1}{T}(\mathscr{H}_0-\mathscr{F}_0)\right), \tag{5.6}$$

$$\mathscr{F}_0 = -T\log\int \exp\left(-\frac{1}{T}\mathscr{H}_0\right) \mathrm{d}X = -T\log(2\pi T)^{L/2} \tag{5.7}$$

We need to minimize $\mathscr{F}_{MDS}(P^0) = <\mathscr{H}_{MDS}-\mathscr{H}_0> + \mathscr{F}_0(P^0)$ with respect to μ_i. Since $-<\mathscr{H}_0> + \mathscr{F}_0(P^0)$ is independent to μ_i, only $<\mathscr{H}_{MDS}>$ part is necessary to be minimized with regard to μ_i. If we apply

$< x_i x_i >= \mu_i \mu_i + TL$ to $< \mathscr{H}_{MDS} >$, then $< \mathscr{H}_{MDS} >$ can be deployed as following:

$$< \mathscr{H}_{MDS} > = \sum_{i<j\leq N}^{N} w_{ij}(< \|x_i - x_j\| > -\delta_{ij})^2 \tag{5.8}$$

$$= \sum_{i<j\leq N}^{N} w_{ij}(\sqrt{\|\mu_i - \mu_j\|^2 + 2TL} - \delta_{ij})^2 \tag{5.9}$$

$$\approx \sum_{i<j\leq N}^{N} w_{ij}(\|\mu_i - \mu_j\| + \sqrt{2TL} - \delta_{ij})^2 \tag{5.10}$$

where $\|a\|$ is Norm_2 of a vector a. Eq. (5.9) can be approximated to Eq. (5.10), since the bigger T, the smaller $\|\mu_i - \mu_j\|$ and the smaller T, the bigger $\|\mu_i - \mu_j\|$.

In [41], Klock and Buhmann tried to find an approximation of $P^G(X)$ with a mean field factorization method by minimizing the Kullback-Leibler (KL) divergence using an EM approach. The found parameters obtained by minimizing the KL-divergence between $P^G(X)$ and $P^0(X)$ using the EM approach are essentially the expected mapping in the target dimension under the current problem space with computational temperature (T).

In contrast, we try to find expected mapping, which minimizes $\mathscr{F}_{MDS}(P^0)$, directly with a new objective function ($\hat{\sigma}$); this which is then applied using a DA approach to the MDS problem space with a computational temperature (T) by well-known EM-like MDS solution, called SMACOF [20]. Therefore, as T varies, the problem space also varies, and the SMACOF algorithm is used to find expected mapping under each problem space at a corresponding T. In order to apply SMACOF algorithm to DA method, we substitute the original STRESS equation (5.1) with Eq. (5.10). Note that μ_i and μ_j are the expected mappings we are looking for, so we can consider $\|\mu_i - \mu_j\|$ as $d_{ij}(X_T)$, where 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 as follows:

$$\hat{\sigma} = \sum_{i<j\leq N}^{N} w_{ij}(d_{ij}(X_T) + \sqrt{2TL} - \delta_{ij})^2 \tag{5.11}$$

$$= \sum_{i<j\leq N}^{N} w_{ij}(d_{ij}(X_T) - \hat{\delta}_{ij})^2 \tag{5.12}$$

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

$$\hat{\delta}_{ij} = \begin{cases} \delta_{ij} - \sqrt{2TL} & \text{if } \delta_{ij} > \sqrt{2TL} \\ 0 & \text{otherwise} \end{cases} \tag{5.13}$$

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. (5.13) can be written as follows:

$$\hat{\delta}_{ij} = \begin{cases} \delta_{ij} - T\sqrt{2L} & \text{if } \delta_{ij} > T\sqrt{2L} \\ 0 & \text{otherwise.} \end{cases} \tag{5.14}$$

Now, we can apply SMACOF to find expected mapping with respect to the new STRESS (5.12) which is based on computational temperature T. 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 $\mathscr{F}$ as in Eq. (2.22). Generally, the DA approach starts with a high T and gets cooled down T as time goes on, like the physical annealing process. However, if the starting computational temperature (T_0) is very high, a condition which results in all $\hat{\delta}_{ij}$ becoming ZERO, then all points will be mapped at origin (O). Once all mappings are at the origin, then the Guttman transform is unable to construct other mappings except the mapping of all at the origin, since the Guttman transform does multiplication iteratively with previous mapping to calculate current mapping. Thus, we need to calculate T_0 which makes at least one $\hat{\delta}_{ij}$ bigger than ZERO, so that at least one of the points is not located at O.

With computed T_0, the $\widehat{\Delta}_0 = [\hat{\delta}_{ij}]$ can be calculated, and we are able to run SMACOF algorithm with

Algorithm 5.1 DA-SMACOF algorithm

Input: Δ and α /* $0 < \alpha < 1$ */

1: Compute T_0
2: Compute $\widehat{\Delta}_0 = [\hat{\delta}_{ij}]$ based on Eq. (5.14).
3: Generate random initial mapping X_0.
4: $k \Leftarrow 0$;
5: **while** $T_k \geq T_{min}$ **do**
6: X_{k+1} = output of SMACOF with $\widehat{\Delta}_k$ and X_k. 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 $\widehat{\Delta}_{k+1}$ w.r.t. T_{k+1}.
9: $k \Leftarrow k+1$;
10: **end while**

/* Finally, we will run SMACOF with original dissimilarity matrix (Δ) by using X_k as the initial mapping. */

11: X = output of SMACOF based on Δ and X_k.
12: **return:** X;

respect to Eq. (5.12). After a new mapping is generated with T_0 by the SMACOF algorithm, say X_0, then we will cool down the temperature in an exponential way, like $T_{k+1} = \alpha T_k$, and keep completing the 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 STRESS (5.1). We will assume a uniform weight $\forall w_{ij} = 1$ where $0 < i < j \leq N$. The proposed deterministic annealing SMACOF algorithm, called DA-SMACOF, is illustrated in Algorithm 5.1.

5.3.1 Temperature Cooling Mechanism

Although we introduced the DA-SMACOF algorithm with an *exponential* cooling mechanism for cooling computational temperature ($T_{k+1} = \alpha T_k$) where $0 < \alpha < 1$, we can also extend DA-SMACOF with a *linear* cooling mechanism for decreasing the computational temperature, which is illustrated as follows: $T_{k+1} = T_k - \beta$ where β is a small constant value. In contrast to those fixed cooling mechanisms (exponential and linear mechanism) which are not cooling adaptively based on the current status of the solution, Choi et al. [17] recently proposed an adaptive cooling method related to the DA approach for Generative Topographic

Mapping (GTM) [11, 12].

Since DA methods are highly dependent on the movement of the computational temperature, it is worthwhile to investigate different cooling mechanism, such as exponential and linear cooling scheme. For implementation of both cooling mechanism, we calculate T_{max} based on the given original dissimilarity matrix (Δ) and set T_{min} as T_{max} multiplied by a small number, i.e. 0.01 or 0.001. For the exponential cooling mechanism, we can simply multiply α on T until $T \leq T_{min}$. On the other hand, we need to calculate β, for the linear cooling scheme, based on user defined *step number* (s) as well as T_{max} and T_{min} as in Eq. (5.15):

$$\beta = \frac{T_{max} - T_{min}}{s}. \tag{5.15}$$

Figure 5.1 depicts computational temperature movement examples of exponential cooling mechanism and linear cooling mechanism with real data used in Section 5.5.3. In Figure 5.1, we used parameters for the cooling method as follows: $T_{min} = 0.01 \times T_{max}$ and $s = 100$ for a linear cooling scheme.

5.4 Experimental Analysis

For the performance analysis of the proposed deterministic annealing MDS algorithm, called **DA-SMACOF**, we would like to examine DA-SMACOF's capability of avoiding local optima in terms of its objective function value (normalized STRESS in (5.16)) and the sensitivity of the initial configuration by comparing it with the original EM-like SMACOF algorithm and MDS by Distance Smoothing [35] (**MDS-DistSmooth** hereafter for short) which tries to find global optimum mapping. We have tested the above algorithms with many different data sets, including well-known benchmarking data sets from the UCI machine learning repository[1] as well as some real application data, such as the chemical compound data and biological sequence data, in order to evaluate the proposed DA-SMACOF.

[1] UCI Machine Learning Repository, `http://archive.ics.uci.edu/ml/`

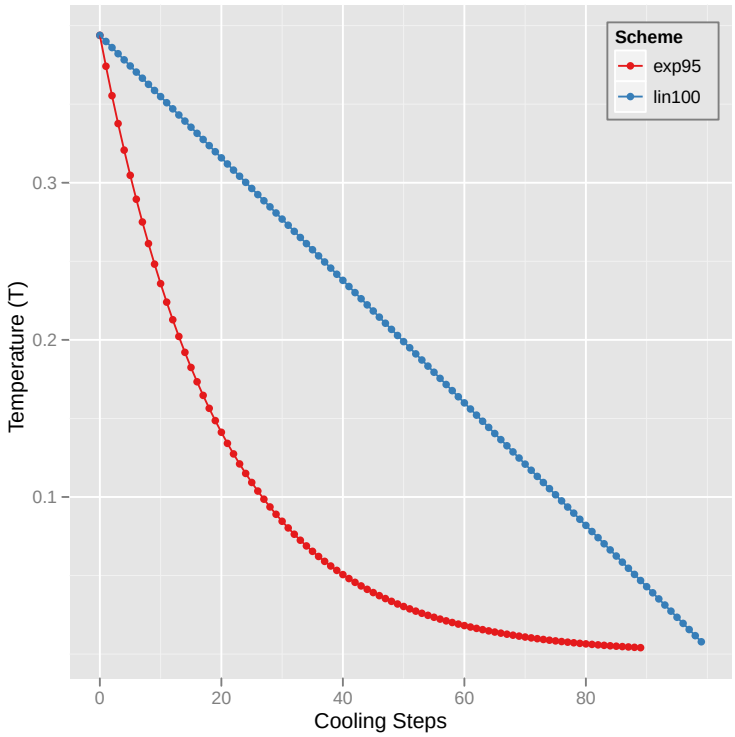


Figure 5.1: The computational temperature movement with respect to two different cooling temperature mechanisms (**exponential** and **linear**).

Since the MDS-DistSmooth requires a number of smoothing steps which affects the degree of smoothness, and the cooling parameter (α) of computational temperature (T) affects the annealing procedure in DA-SMACOF, we examine two different numbers of smoothing step numbers ($s = 100$ and $s = 200$) for the MDS-DistSmooth and three different cooling parameters ($\alpha = 0.9$, 0.95, and 0.99) for the DA-SMACOF algorithm, as well. (Hereafter, MDS-DistSmooth with smoothing steps $s = 100$ and $s = 200$ are described by **DS-s100** and **DS-s200** respectively, and the DA-SMACOF with temperature cooling parameters $\alpha = 0.9$, 0.95, and 0.99 are represented by **DA-exp90**, **DA-exp95**, and **DA-exp99**, correspondingly.) We also examine two different thresholds for the stopping condition, i.e. $\varepsilon = 10^{-5}$ and $\varepsilon = 10^{-6}$, for tested algorithms.

To compare the mapping quality of the proposed DA-SMACOF with SMACOF and MDS-DistSmooth,

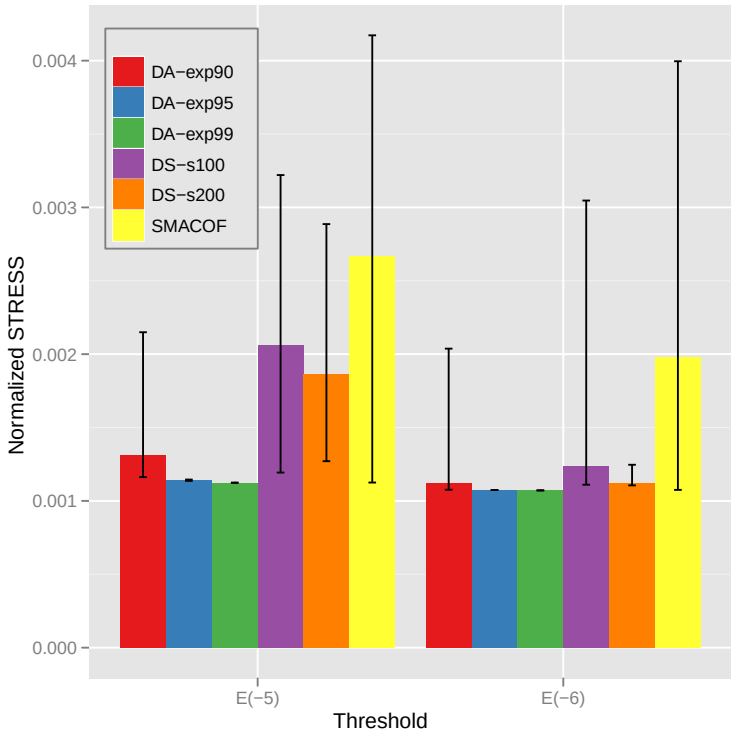


Figure 5.2: The normalized STRESS comparison of the **iris** data mapping results in 2D space. The bar graph illustrates the average of 50 runs with random initialization, and the corresponding error bar represents the minimum and maximum of the normalized STRESS values of SMACOF, MDS-DistSmooth with different smoothing steps ($s = 100$ and $s = 200$) (**DS-s100** and **-s200** hereafter for short), and DA-SMACOF with different cooling parameters ($\alpha = 0.9$, 0.95, and 0.99) (**DA-exp90,-exp95**, and **-exp99** hereafter for short). The x-axis is the threshold value for the stopping condition of iterations (10^{-5} and 10^{-6}).

we measure the normalized STRESS which substitutes w_{ij} in (5.1) for $1/\sum_{i<j}\delta_{ij}^2$ like in the following:

$$\sigma(X) = \sum_{i<j\leq N} \frac{1}{\sum_{i<j}\delta_{ij}^2}(d_{ij}(X) - \delta_{ij})^2 \tag{5.16}$$

in that the normalized STRESS value denotes the relative portion of the squared distance error rates of the given data set without regard to the scale of δ_{ij}.

5.4.1 Iris Data

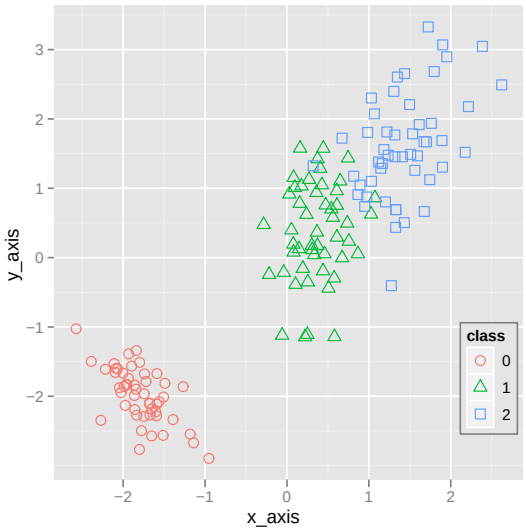
The iris data[2] set is a very well-known benchmarking data set for the data mining and pattern recognition communities. Each data item consists of four different real values (a.k.a. *4D real-valued vector*) and each value represents an attribute of each instance, such as length or width of sepal (or petal). There are three different classes (*Iris Setosa, Iris Versicolour, and Iris Virginica*) in the iris data set and each class contains 50 instances, so there are total 150 instances in the iris data set. It is known that one class is linearly separable from the other two, but the remaining two are not linearly seperable from each other.

In Figure 5.2, The mapping quality of the constructed configurations of the iris data by SMACOF, MDS-DistSmooth, and DA-SMACOF is compared by the average, the minimun, and the maximum of the normalized STRESS values among 50 random-initial runnings. The proposed DA-SMACOF with all tested cooling parameters, including quite fast cooling parameter ($\alpha = 0.9$), outperforms SMACOF and MDS-DistSmooth in Figure 5.2 except DS-s200 case with $\varepsilon = 10^{-6}$. Although DS-s200 with $\varepsilon = 10^{-6}$ is comparable to the DA-SMACOF results, DS-s200 takes almost 3 times longer than DA-exp95 with $\varepsilon = 10^{-6}$ which shows more consistent results than DS-s200.

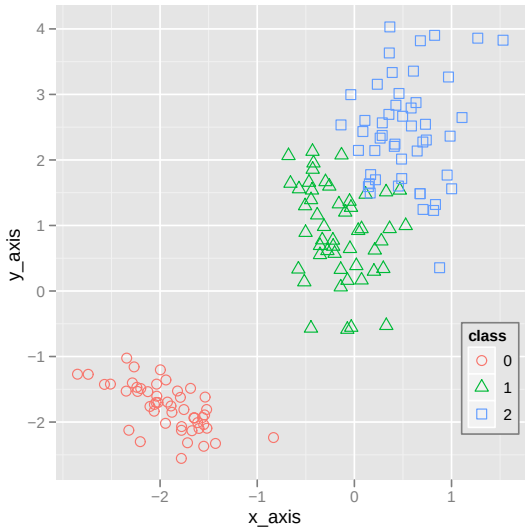
Numerically, DA-exp95 improves the mapping quality of 57.1% and 45.8% of the SMACOF results in terms of the average of the STRESS values with $\varepsilon = 10^{-5}$ and $\varepsilon = 10^{-6}$, correspondingly. DA-exp95 shows better mapping quality – about 43.6% and 13.2% – than even DS-s100, which is the algorithm to find the global optimum, with $\varepsilon = 10^{-5}$ and $\varepsilon = 10^{-6}$.

In terms of sensitivity to the initial configuration, the SMACOF shows very divergent STRESS value distributions for both $\varepsilon = 10^{-5}$ and $\varepsilon = 10^{-6}$ cases in Figure 5.2. This illustrates that SMACOF is quite sensitive to the initial configuration (a.k.a. easy to be trapped in *local optima*). In addition, MDS-DistSmooth also shows a relatively high sensitivity to the initial configuration with the iris data set although the degree

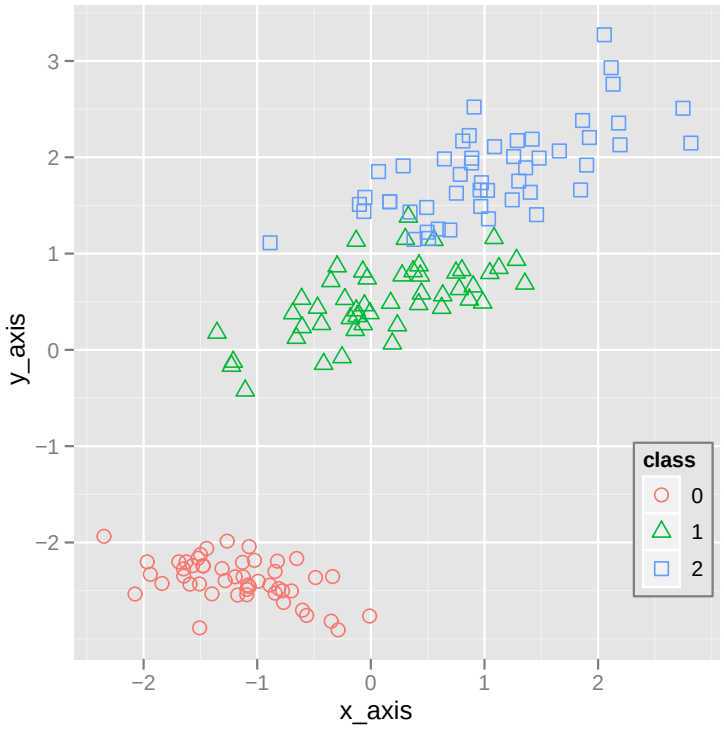
[2] Iris Data set, `http://archive.ics.uci.edu/mi/datasets/Iris`



(a) Iris by EM Median



(b) Iris by DS Median



(c) Iris by DA Median

Figure 5.3: The 2D median output mappings of iris data with SMACOF (a), DS-s100 (b), and DA-exp95 (c), whose threshold value for the stopping condition is 10^{-5}. Final normalized STRESS values of (a), (b), and (c) are 0.00264628, 0.00208246, and 0.00114387, correspondingly.

of divergence is less than SMACOF algorithm. In contrast to other algorithms, the proposed DA-SMACOF shows high consistency without regard to initial setting which we could interprete as it being likely to avoid local optima. Since it is well-known that a slow cooling temperature is necessary to avoid local optima, we expected that the DA-exp90 might be trapped in local optima as shown in Figure 5.2. Although the DA-exp90 cases show some variations, the DA-exp90 cases still show much better results than the SMACOF and the MDS-DistSmooth except for the DS-s200 with a $\varepsilon = 10^{-6}$ case. In fact, the standard deviation of DA-exp95 with the $\varepsilon = 10^{-5}$ result is 1.08×10^{-6} and DA-exp99 with $\varepsilon = 10^{-5}$ and DA-exp95/exp99 with $\varepsilon = 10^{-6}$ shows a ZERO standard deviation in terms of the STRESS values of 50 random-initial runs. We can also note that the difference of the DA-SMACOF results between $\varepsilon = 10^{-5}$ and $\varepsilon = 10^{-6}$ is negligible with the iris data, whereas the average of SMACOF and MDS-DistSmooth (DS-s100) with $\varepsilon = 10^{-5}$ is about 35.5% and 81.6% worse than corresponding $\varepsilon = 10^{-6}$ results.

Figure 5.3 illustrates the difference of actual mapping SMACOF, MDS-DistSmooth, and DA-SMACOF. All of the mappings are the median results of a stopping condition with 10^{-5} threshold value. The mapping in Figure 5.3a is the 2D mapping result of the median valued SMACOF, and Figure 5.3b represents the median result of the MDS-DistSmooth. Three mappings in Figure 5.3 are a little bit different from one another, and clearer structure differentiation between class 1 and class 2 is shown in Figure 5.3c which is the median STRESS valued result of DA-SMACOF.

5.4.2 Chemical Compound Data

The second data set consists of chemical compound data with 333 instances represented by 155 dimensional real-valued vectors. For the given original dissimilarity (Δ), we measure the Euclidean distance of each instance of pairs based on feature vectors as well as the iris data set.

Figure 5.4 depicts the average mapping quality of 50 runs of 333 chemical compounds mapping results with regard to different experimental setups as in the above description. For the chemical compound

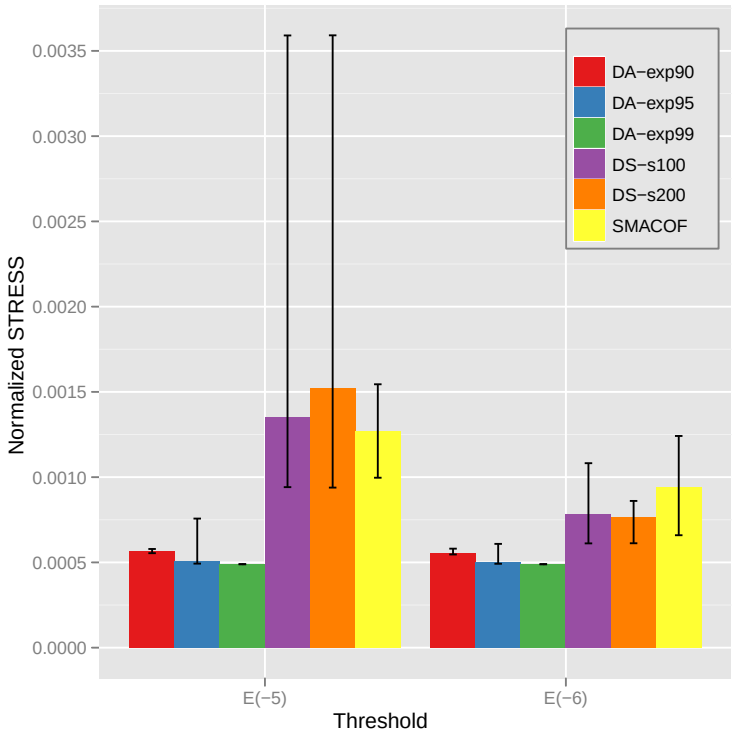


Figure 5.4: The normalized STRESS comparison of the **chemical compound** data mapping results in 2D space. The bar graph illustrates the average of 50 runs with random initialization, and the corresponding error bar represents the minimum and maximum of the normalized STRESS values of SMACOF, DS-s100 and -s200, and DA-exp90, DA-exp95, and DA-exp99. The x-axis is the threshold value for the stopping condition of iterations (10^{-5} and 10^{-6}).

data set, all the experimental results of the proposed DA-SMACOF (DA-exp90, DA-exp95, and DA-exp99) show superior performance to the SMACOF and the MDS-DistSmooth with both $\varepsilon = 10^{-5}$ and $\varepsilon = 10^{-6}$ stopping conditions. In detail, the average STRESS of the SMACOF is 2.50 and 1.88 times larger than corresponding DA-SMACOF results with $\varepsilon = 10^{-5}$ and $\varepsilon = 10^{-6}$ threshold, and the average STRESS of the MDS-DistSmooth shows 2.66 and 1.57 times larger than the DA-SMACOF algorithm with $\varepsilon = 10^{-5}$ and $\varepsilon = 10^{-6}$. Furthermore, the minimum STRESS values of SMACOF and MDS-DistSmooth experiments are larger than the average of all DA-SMACOF results. One interesting phenomena in Figure 5.4 is that the MDS-DistSmooth shows worse performance on average with $\varepsilon = 10^{-5}$ stopping condition than SMACOF and DS-s100 shows better than DS-s200.

Similar to Figure 5.2, all the SMACOF and the MDS-DistSmooth experimental results show a higher divergence in terms of the STRESS values in Figure 5.4 than the proposed DA-SMACOF. On the other hand, DA-SMACOF shows much less divergence with respect to the STRESS values, especially in the DA-exp99 case.

For the comparison between different cooling parameters, as we expected, the DA-exp90 shows some divergence and a little bit higher average than the DA-exp95 and DA-exp99, but much less of an average than the SMACOF. Interestingly, DA-exp95 shows relatively larger divergence than DA-exp90 due to outliers. However, those outliers of DA-exp95 happened rarely among 50 runs and most of DA-exp95 running results are similar to the minimum value of the corresponding test.

5.4.3 Cancer Data

The cancer data[3] set is a well-known data set found in the UCI Machine Learning Repository. Each data item consists of 11 columns and the first and the last column represent the *id-number* and the *class* correspondingly. The remaining 9 columns are attribute values described in integers ranging from 1 to 10. There are two different classes (*benign and malignant*) in the cancer data set. Originally, it contained 699 data item; however, we used only 683 data points which have every attribute value included, since 16 items have some missing information.

Figure 5.5 depicts the average mapping quality of 50 runs for 683 cancer data mapping results with regard to different experimental setups as in the above. For the cancer data set, all experimental results of the proposed DA-SMACOF (DA-exp90, DA-exp95, and DA-exp99) show superior performance to SMACOF and MDS-DistSmooth with $\varepsilon = 10^{-5}$, and better than SMACOF and comparable to MDS-DistSmooth with $\varepsilon = 10^{-6}$ stopping conditions. Interestingly, the DA-exp99 case shows worse results than the DA-exp95

[3]Breast Cancer Data set, `http://archive.ics.uci.edu/mi/datasets/Breast+Cancer+Wisconsin+(Original)`

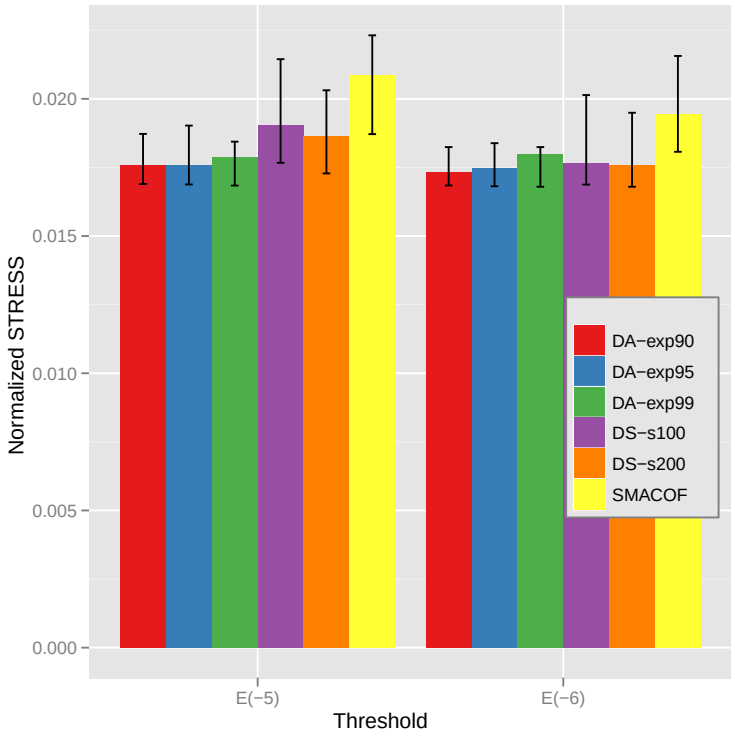


Figure 5.5: The normalized STRESS comparison of the **breast cancer** data mapping results in 2D space. The bar graph illustrates the average of 50 runs with random initialization, and the corresponding error bar represents the minimum and maximum of the normalized STRESS values of SMACOF, DS-s100 and -s200, and DA-exp90,DA-exp95, and DA-exp99. The x-axis is the threshold value for the stopping condition of iterations (10^{-5} and 10^{-6}).

and the DA-exp90 results, although the DA-exp99 case find the most minimum mapping in terms of normalized STRESS value. In detail, the average STRESS of SMACOF is 18.6% and 11.3% worse than the corresponding DA-SMACOF results with $\varepsilon = 10^{-5}$ and $\varepsilon = 10^{-6}$ threshold, and the average STRESS of the MDS-DistSmooth shows a performance which is 8.3% worse than the DA-SMACOF with $\varepsilon = 10^{-5}$ and comparable to the DA-SMACOF with $\varepsilon = 10^{-6}$.

Although the DA-SMACOF experimental results show some divergence in terms of the STRESS values in Figure 5.5, in contrast to Figure 5.2 and Figure 5.4, the DA-SMACOF experimental results show less divergence of the STRESS values than the SMACOF and the MDS-DistSmooth in Figure 5.5.

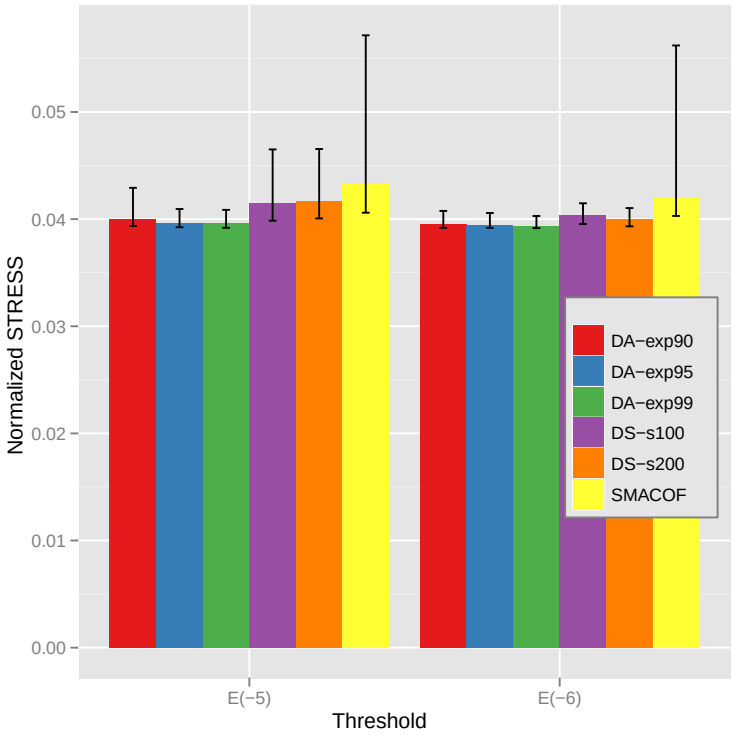


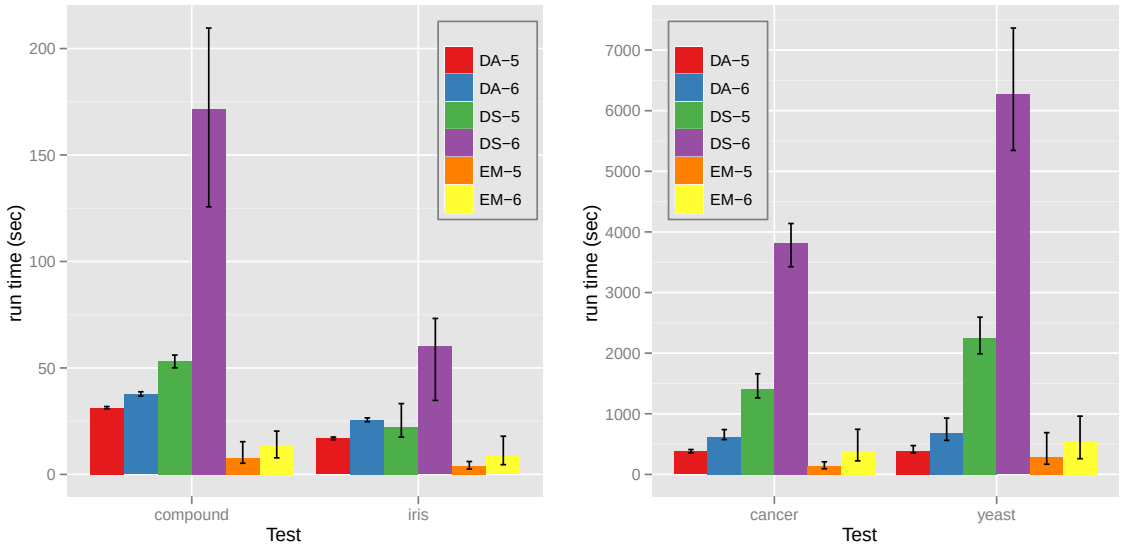
Figure 5.6: The normalized STRESS comparison of the **yeast** data mapping results in 2D space. The bar graph illustrates the average of 50 runs with random initialization, and the corresponding error bar represents the minimum and maximum of the normalized STRESS values of SMACOF, DS-s100 and -s200, and DA-exp90,DA-exp95, and DA-exp99. The x-axis is the threshold value for the stopping condition of iterations (10^{-5} and 10^{-6}).

5.4.4 Yeast Data

The yeast data[4] set is composed of 1484 entities, and each entity is represented by 8 real-value attributes in addition to the sequence name and class labels.

The normalized STRESS comparison of the yeast mapping results by different algorithms is illustrated in Figure 5.6 in terms of the average mapping quality of 50 runs for 1484 points mapping. DA-SMACOF shows better performance than the other two algorithms in this experiments similar to the above experiments. The SMACOF keeps showing a much higher divergence rather than the DA-SMACOF with both stopping

[4]Yeast Data set, `http://archive.ics.uci.edu/mi/datasets/Yeast`



(a) Small Data Runtime

(b) Large Data Runtime

Figure 5.7: The average running time comparison between SMACOF, MDS-DistSmooth ($s = 100$), and DA-SMACOF (DA-exp95) for 2D mappings with tested data sets. The error bar represents the minimum and maximum running time. **EM-5/EM-6** represents SMACOF with $10^{-5}/10^{-6}$ threshold, and **DS-5/DS-6** and **DA-5/DA-6** represents the runtime results of MDS-DistSmooth and DA-SMACOF, correspondingly, in the same way.

condition cases. Also, the MDS-DistSmooth shows divergent STRESS distribution with the $\varepsilon = 10^{-5}$ stopping condition, but not with the $\varepsilon = 10^{-6}$ stopping condition. The DA-SMACOF shows quite stable results except for the DA-exp90 case with a $\varepsilon = 10^{-5}$ stopping condition, as well as a better solution. In terms of the best mapping (a.k.a. minimum normalized STRESS value), all DA-SMACOF experiments obtain a better solution than the SMACOF and MDS-DistSmooth, and even best result of the SMACOF is worse than the average of the proposed DA approach.

5.4.5 Running Time Comparison

From Section 5.4.1 to Section 5.4.4, we have been analyzing the mapping quality by comparing DA-SMACOF with SMACOF and MDS-DistSmooth. DA-SMACOF outperforms SMACOF in all test cases, and

outperforms or is comparable to MDS-DistSmooth. In this section, we would like to compare the running time among those algorithms. Figure 5.7 describes the average running time of each test case for SMACOF, MDS-DistSmooth, and DA-exp95 with 50 runs for the tested data. In order to make a distinct landscape in Figure 5.7, we plot the quadrupled runtime results of the **iris** and **cancer** data.

In Figure 5.7, all runnings are performed in sequential computing with AMD Opteron 8356 2.3GHz CPU and 16GB memory. As shown in Figure 5.7, DA-SMACOF is a few times slower than SMACOF but faster than MDS-DistSmooth in all test cases. In detail, DA-SMACOF takes 2.8 to 4.2 times longer than SMACOF but 1.3 to 4.6 times shorter than the MDS-DistSmooth with the **iris** and **compound** data set in Figure 5.7a. Also, DA-SMACOF takes 1.3 to 2.8 times longer than the SMACOF but 3.7 to 9.1 times shorter than the MDS-DistSmooth with the **cancer** and **yeast** data set in Figure 5.7b. Interestingly, less deviation is shown by DA-SMACOF than other compared algorithms in all cases, with respect to running time as well as STRESS values.

5.5 Experiment Analysis of Large Data Sets

So far, the experimental data sets are of a relatively small size, i.e. 150, 333, 683 and 1484 instances, in Section 5.4. In contrast to smaller data sizes as in the above tests, I present an experimental analysis for larger data sets containing anywhere from 30,000 points (sequences) to 100,000 points. Since MDS algorithms requires $\mathcal{O}(N^2)$ of main memory, we have to use much larger amounts of memory than main memory in a single node for running analyseswith 30,000 or more points. Thus, we use the distributed memory version of the SMACOF (and DA-SMACOF) algorithms [16] to run these large data sets. Also, we experiment dimension reduction in three dimensional space (**3D**) as well as 2D space, since the configuration of 2D mapping might involve high constraints for a larger data set.

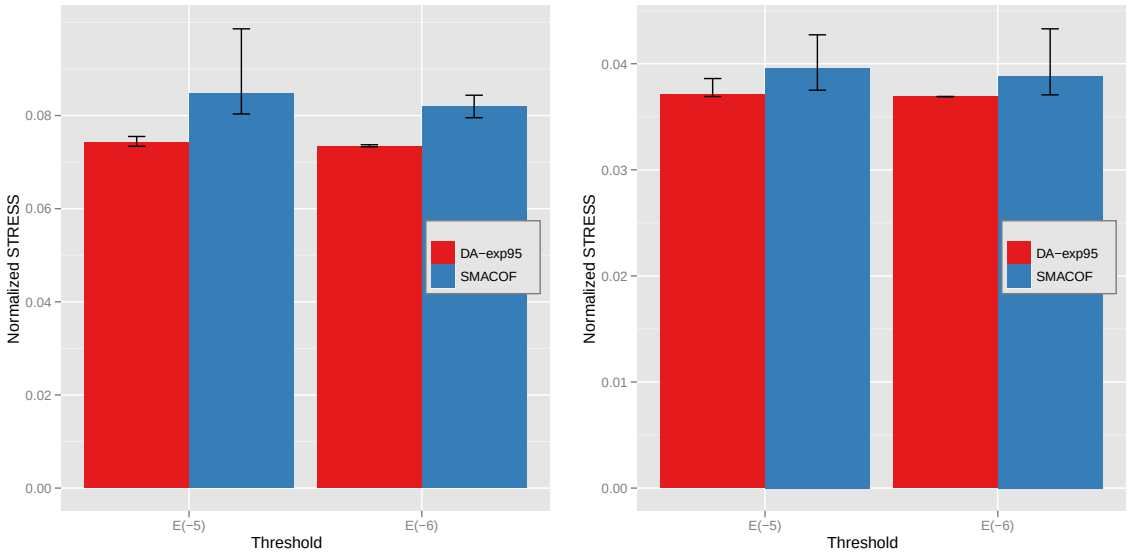
Since the running time of the SMACOF (and DA-SMACOF) for larger data sets is much longer than that

for smaller data sets, we would compare the SMACOF result with those of the DA-SMACOF with $\alpha = 0.95$ for the exponential cooling scheme (**DA-exp95**) based on previous experimental comparisons among DA-exp90, DA-exp95, DA-exp99 in Section 5.4. Instead of comparing three different exponential cooling parameters (α), we would like to explore other DA parameters, such as a *linear cooling scheme* for computational temperature (T) cooling scheme and a *hybrid stop condition* which uses two different stop condition parameters ($\varepsilon_1, \varepsilon_2$) for the annealing period (where $T > 0$) and after the annealing period (where $T = 0$). For instance, we use $\varepsilon_1 = 10^{-5}$ when $T > 0$ and set $\varepsilon_2 = 10^{-6}$ when $T = 0$ for both exponential and linear cooling schemes in this section.

The rationale of using the hybrid stop condition is highly related to our previous experimental results in which the DA-SMACOF results with $\varepsilon = 10^{-5}$ were highly similar to those with $\varepsilon = 10^{-6}$. Based on the similarity of the DA-SMACOF results with different stop condition parameters, it is a logical deduction that the DA-SMACOF is able to avoid local optima if the iteration stop condition parameter (ε) is small enough to reach an approximated average mappings at each computation temperature (T) but is not necessarily too small to avoid local optima. The small difference in the STRESS values between DA-SMACOF with $\varepsilon = 10^{-5}$ and with $\varepsilon = 10^{-6}$ might result from the nature of different stop conditions when $T = 0$. In terms of running time, however, the iteration stop condition parameter (ε) makes a significant difference between when $\varepsilon = 10^{-5}$ and $\varepsilon = 10^{-6}$. Thus, if we could make up the small difference of the STRESS values between $\varepsilon = 10^{-5}$ and $\varepsilon = 10^{-6}$ by using the hybrid stop condition approach, it would provide a considerable gain with respect to running time.

5.5.1 Metagenomics Data

One of the large data sets we used for evaluation of the DA-SMACOF algorithm is a biological sequence data with respect to the metagenomics study. Although it is hard to present a biological sequence in a feature vector, people can calculate a dissimilarity value between two different sequences by using pairwise sequence



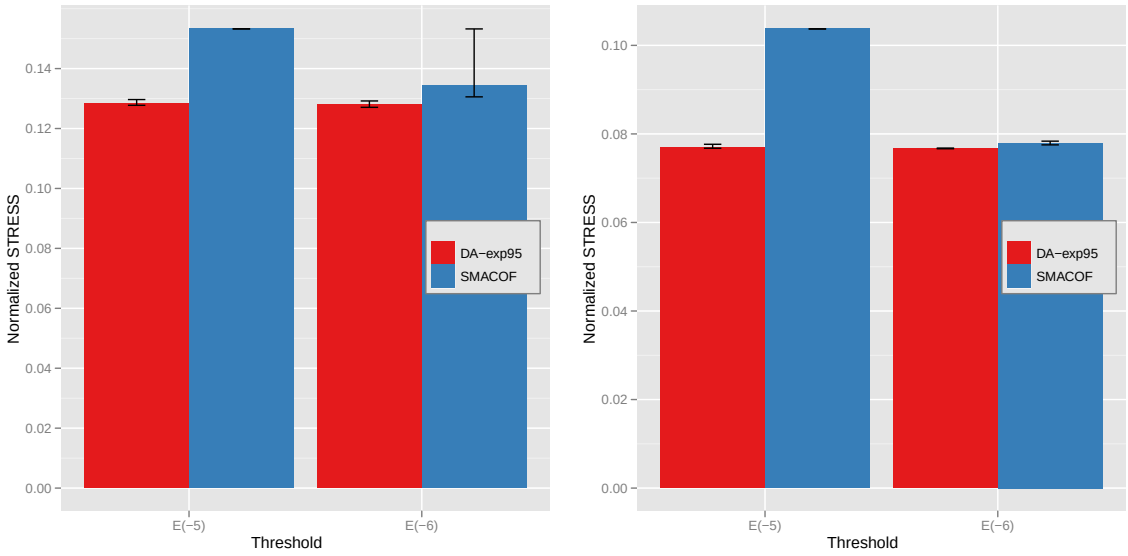
(a) MC30000 2D STRESS

(b) MC30000 3D STRESS

Figure 5.8: The normalized STRESS comparison of the **metagenomics sequence** data mapping results in 2D and 3D space. The 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 values of SMACOF and DA-SMACOF with $\alpha = 0.95$. The x-axis is the threshold value for the iteration stopping condition of the SMACOF and DA-SMACOF algorithms (10^{-5} and 10^{-6}).

alignment algorithms, like the Smith Waterman - Gotoh (*SW-G*) algorithm [30, 65] which we used for this metagenomics data set.

Figure 5.8 is the comparison between the average of 10 random initial runs of DA-SMACOF (DA-exp95) and SMACOF with the metagenomics data set, which contains 30000 points. Similar to previous results, the SMACOF shows a tendency to be trapped in local optima by depicting some variation and larger STRESS values, and even the minimum STRESS values are bigger than any results of DA-exp95 in 2D mapping case. For 2D mapping, the DA-exp95 results actually demonstrate a 12.6% and 10.4% improvement compared to the SMACOF on average, with $\varepsilon = 10^{-5}$ and $\varepsilon = 10^{-6}$, correspondingly. The DA-exp95 results in 3D space are 6.3% and 5.0% better than the SMACOF in average, with $\varepsilon = 10^{-5}$ and $\varepsilon = 10^{-6}$, correspondingly, as well. Note that the STRESS values of the mappings in 3D space are much less than those of the mappings in 2D space, regardless of algorithms.



(a) ALU50058 2D STRESS

(b) ALU50058 3D STRESS

Figure 5.9: The normalized STRESS comparison of the **ALU sequence** data mapping results in 2D and 3D space. The 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 values of SMACOF and DA-SMACOF with $\alpha = 0.95$. The x-axis is the threshold value for the iteration stopping condition of the SMACOF and DA-SMACOF algorithms (10^{-5} and 10^{-6}).

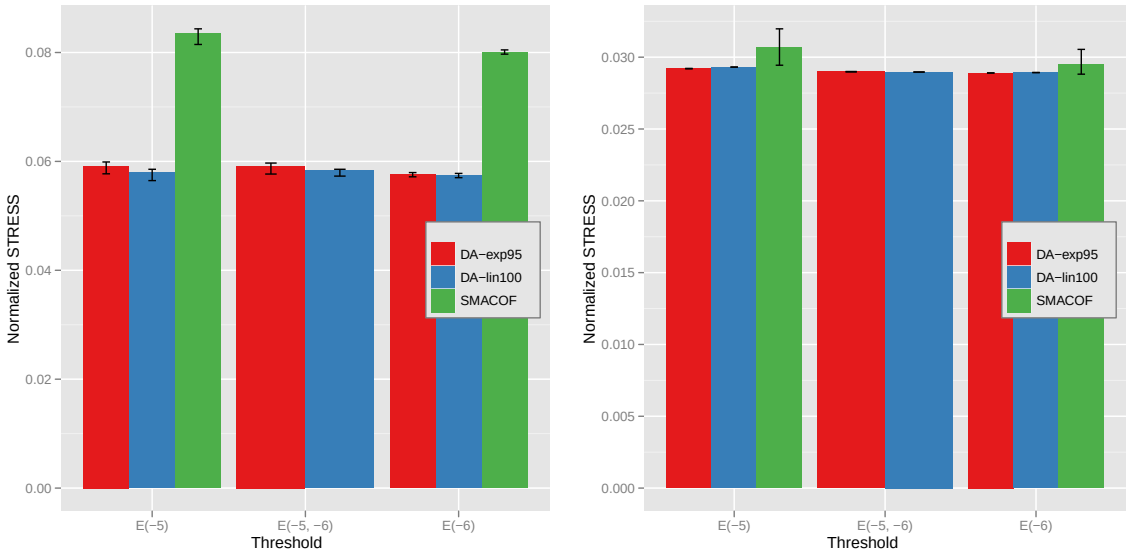
As shown in Figure 5.8, all of the DA-exp95 results are very similar to each other, especially when the stopping condition is $\varepsilon = 10^{-6}$. Although DA-exp95 with $\varepsilon = 10^{-5}$ in 3D shows some variation in Figure 5.8-(b), in fact, all the mappings except the maximum STRESS case show very similar STRESS values as depicted, in that the minimum STRESS value is close to the average. Furthermore, all the generated mappings by DA-exp95 with $\varepsilon = 10^{-6}$ in 3D case shows the same STRESS value up to the 10^{-7} scale. In contrast to the DA-SMACOF, the SMACOF shows a larger variation in both stopping conditions in Figure 5.8, and all the mappings generated by the SMACOF are worse than all the DA-exp95 results except the outlier (the maximum STRESS case with 10^{-5} stop condition).

5.5.2 ALU Sequence Data

We also used another type of biological sequence data, which consisted of 50058 samples of Alu repeats [9] coming from the Human and Chimpanzee genomes, for the purpose of comparison between the proposed DA-SMACOF and SMACOF. The required all-pairwise dissimilarity matrix for MDS problem is also computed based on pairwise sequence alignment results as in Section 5.5.1.

In Figure 5.9, the comparison between the average of 10 random initial runs of DA-SMACOF (DA-exp95) and SMACOF with 50058 ALU sequence data in 2D and 3D with two different iteration stop condition is illustrated. Similar to other results, in every experiment, the DA-exp95 shows better quality of mappings in terms of the average normalized STRESS values than do the SMACOF results. As shown in Figure 5.9, the maximum normalized STRESS values of all DA-exp95 experiments are less than the minimum normalized STRESS value of corresponding SMACOF experiments. In detail, the DA-exp95 results in the 2D mapping show 16.3% and 4.6% improvement, compared to the SMACOF, on average with $\varepsilon = 10^{-5}$ and $\varepsilon = 10^{-6}$, respectively. For 3D mapping, the DA-exp95 results are 25.8% and 1.6% better than the corresponding SMACOF results in average with the same stopping conditions. DA-exp95 also shows a high consistency of mappings with respect to the normalized STRESS values as shown in other examples, and the standard deviation of the normalized STRESS values of DA-exp95 with $\varepsilon = 10^{-6}$ in 3D space is 4.42×10^{-5}.

One interesting feature of the SMACOF result in Figure 5.9 is that both the 2D and 3D SMACOF mappings with the $\varepsilon = 10^{-5}$ stop condition show high consistent normalized STRESS values although these are much worse than the corresponding DA-exp95 results. The main reason for the high consistency of SMACOF with $\varepsilon = 10^{-5}$ is that SMACOF has stopped prematurely due to the relatively large stop condition for the data. As illustrated in Figure 5.15-(b), the runtime of SMACOF with ALU sequences data with $\varepsilon = 10^{-5}$ stop condition is unbelievably faster than the other test with the same data. If we compare the average iteration number, it is extremely clear that the SMACOF results of the ALU data with the $\varepsilon = 10^{-5}$ stop condition have been stopped pre-maturely. The average iteration numbers of the SMACOF with ALU sequence data



(a) 16sRNA 50k 2D STRESS

(b) 16sRNA 50k 3D STRESS

Figure 5.10: The normalized STRESS comparison of the **16s RNA sequence** data with 50000 sequences for mapping in 2D and 3D space. The 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 values of SMACOF, DA-SMACOF with $\alpha = 0.95$ (**DA-exp95**), and DA-SMACOF with linear cooling with 100 steps (**DA-lin100**). The x-axis is the threshold value for the iteration stopping condition of the SMACOF and DA-SMACOF algorithms (10^{-5}, 10^{-6}, and *hybrid* approach).

with $\varepsilon = 10^{-5}$ case are 24.6 (either 24 or 25) and 20.2 (either 20 or 21) iterations for 2D and 3D mapping, correspondingly. These numbers of the SMACOF with the ALU sequence data with $\varepsilon = 10^{-6}$ case, however, are 954.70 and 974.56 iterations with high variations for 2D and 3D mapping, respectively.

5.5.3 16sRNA 50k Data

The third and fourth test data sets are 16s RNA biological sequence data sets with different sizes, such as 50,000 (hereafter **RNA50k**) and 100,000 (hereafter **RNA100k**). Since they are biological sequence data sets, it is hard to represent them with feature vectors but it is possible to generate a pairwise dissimilarity matrix. Instead of the *SW-G* local sequence alignment algorithm [30, 65] which we used to generate a pairwise dissimilarity matrix for the Metagenomics data set, we used the Needleman-Wunsch (NW) algorithm [53],

which is a global sequence alignment algorithm, for creating a pairwise dissimilarity matrix of 16sRNA sequence data sets (both 50k and 100k) for the MDS input. In this section, we also tested a *linear* cooling scheme with 100 steps (hereafter **DA-lin100**) and a *hybrid stop condition* threshold approach as well as a DA-exp95 experimental setting. We experimented mapping in not only 2D space but also 3D space.

Figure 5.10 describes the mapping quality comparison in terms of the average normalized STRESS values based on 10 random runs of each experimental setup with RNA50k data set. In Figure 5.10, DA-SMACOF shows a better quality of mappings in terms of the average STRESS values than the SMACOF results in every experiment, as well. As shown in Figure 5.10-(a), all DA-SMACOF experiments significantly outperform the SMACOF results in 2D space. Numerically, the 2D mapping results of DA-exp95 are 29.1% and 28.0% better than the SMACOF on average with $\varepsilon = 10^{-5}$ and $\varepsilon = 10^{-6}$ stop conditions, correspondingly. For 3D mapping case, the SMACOF shows 5.1% worse than DA-exp95 when $\varepsilon = 10^{-5}$, and 2.1% worse than DA-exp95 when $\varepsilon = 10^{-6}$. As usual, the DA-SMACOF shows consistent mapping quality with little variation of STRESS values for all test cases, especially the mappings in 3D space.

One interesting attribute of the experimental results in Figure 5.10 is that the DA-SMACOF shows a higher consistency in 3D mappings, but SMACOF also shows a higher consistency in 2D mappings (compared to other SMACOF results) with much worse mapping quality compared to DA-SMACOF. However, it is not a result from the premature convergence due to the larger stopping condition as mentioned in Section 5.5.2, since SMACOF with both stopping conditions ($\varepsilon = 10^{-5}, 10^{-6}$) show similarly much higher STRESS values, regardless of stopping conditions. Based on the fact that 2D mapping is much more constrained than 3D mapping, we may interpret the high consistency in 2D mappings of SMACOF with RNA50k data as a result of a deep local optima under the 2D MDS mapping of the RNA50k data problem space.

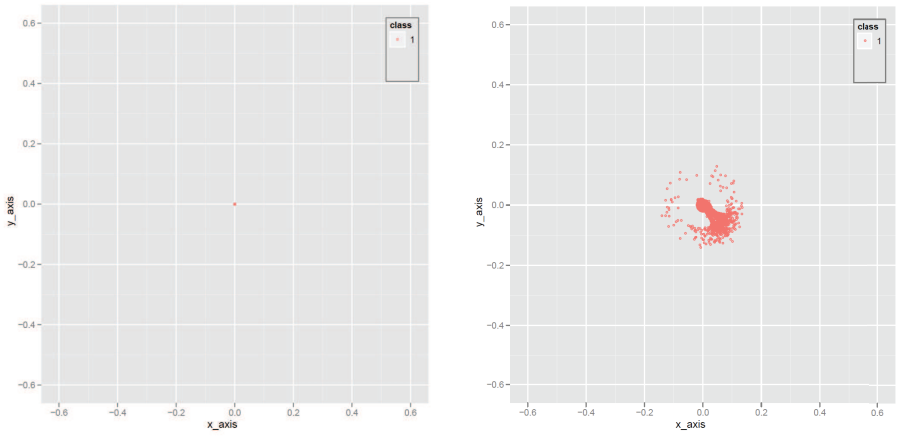
The additional experimental features in Figure 5.10 are a linear cooling scheme and a hybrid stop condition approach. In Figure 5.10, the linear cooling scheme shows slightly better performance in 2D mapping in terms of STRESS values than exponential cooling scheme, and comparable to exponential cooling scheme in

3D mapping results. For the hybrid stop condition approach experiments, the mapping quality of the hybrid stop condition approach is similar to that of the $\varepsilon = 10^{-5}$ case in 2D mappings for both the exponential and linear cooling schemes. The mapping quality of the hybrid approach is similar to that of $\varepsilon = 10^{-6}$ case in the 3D mappings for both the exponential and linear cooling schemes, as opposed to the 2D mappings results. Based on the mapping quality results of the hybrid stop condtion scheme, we infer that the hybrid stop condition scheme works more effectively in less constrained environments. (Here, 3D mapping represents much less constrained environments than 2D mapping.)

Figure 5.11 depicts an example of how the DA-SMACOF forms the mapping in 2D space from the high temperature ($T_0 = \alpha \times T_{max}$) to the final mapping result, and corresponding normalized STRESS value (σ). As illustrated in Figure 5.11-(a), the DA-SMACOF starts from a mapping in which all the points are close to one points whose normalized STRESS value is 0.999995 which is very close to 1.0, as shown in Figure 5.14-(a). As T decreases, the DA-SMACOF algorithm gradually forms a structured mapping from the initial one-point centered mapping. As T is decreased, the corresponding normalized STRESS value (σ) also decreases gradually as in Figure 5.11-(b), and DA-SMACOF forms a little mapping structure as shown in that figure. As the DA-SMACOF algorithm gradually configures a structured mapping as in Figure 5.11-(c),(d), and (e), σ decreases rapidly as small as 0.202537 when $T = 0.103781$, almost five times smaller than when T_0. Finally, the DA-SMACOF algorithm provides the final mapping in 2D space after set $T = 0$ in Figure 5.11-(f). Interestingly, Figure 5.11-(e) and -(f) are configured in very similar shapes with different scales.

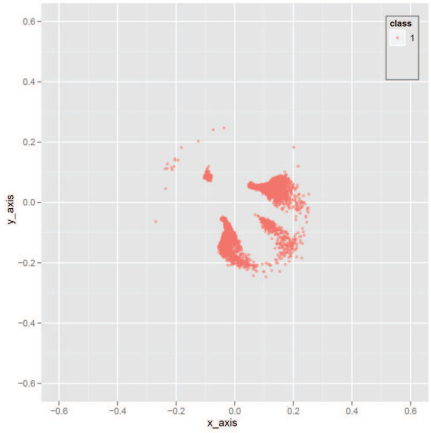
5.5.4 16sRNA 100k Data

The mapping quality comparison in terms of the average normalized STRESS values based on 10 random runs of each experimental setup with **RNA100k** data set is shown in Figure 5.12. Like other experimental results in this chapter, the DA-SMACOF shows a better quality of mappings in terms of the average STRESS values than do the SMACOF results in every experiment. In overall experimental setup, Figure 5.12 shows

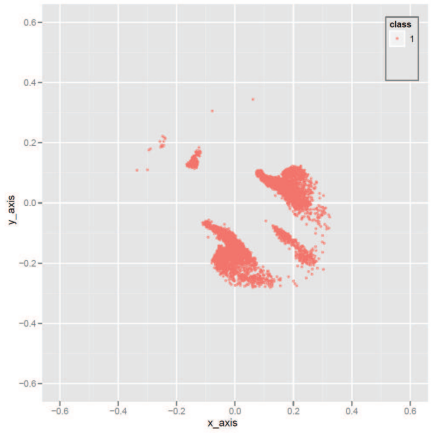


(a) T = 0.374134, σ = 0.999995

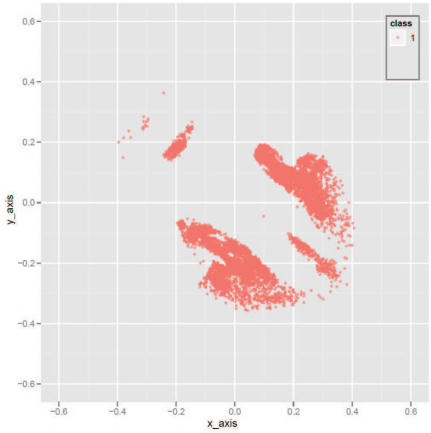
(b) T = 0.275023, σ = 0.926254



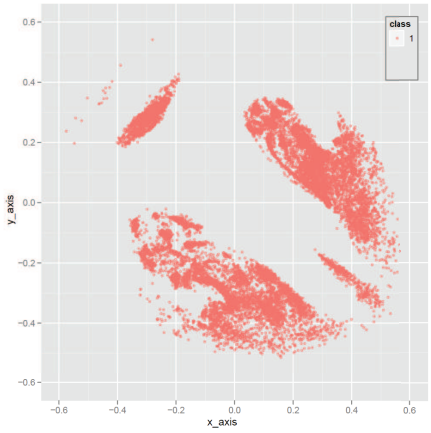
(c) T = 0.202167, σ = 0.562188



(d) T = 0.156433, σ = 0.375549

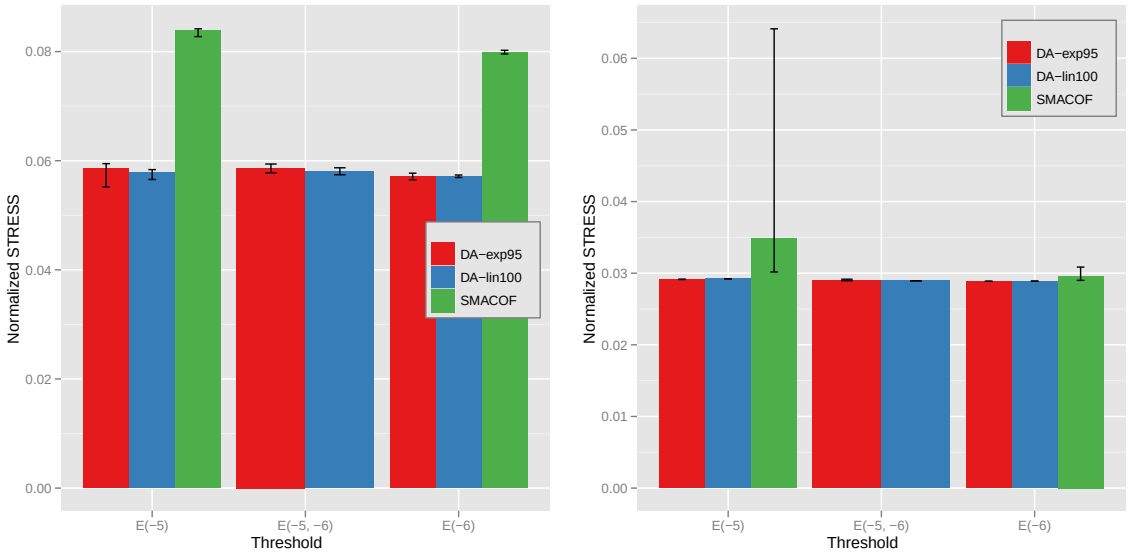


(e) T = 0.103781, σ = 0.202537



(f) T = 0.0, σ = 0.0579866

Figure 5.11: The mapping progress of DA-SMACOF with **RNA50k** data set in 2D space with respect to computational temperature (T) and corresponding normalized STRESS value (σ).



(a) 16sRNA 100k 2D STRESS

(b) 16sRNA 100k 3D STRESS

Figure 5.12: The normalized STRESS comparison of the **16s RNA sequence** data with 100000 sequences for mapping in 2D and 3D space. The 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 values of SMACOF, DA-SMACOF with $\alpha = 0.95$ (**DA-exp95**), and DA-SMACOF with linear cooling with 100 steps (**DA-lin100**). The x-axis is the threshold value for the iteration stopping condition of the SMACOF and DA-SMACOF algorithms (10^{-5}, 10^{-6}, and *hybrid* approach).

a very similar mapping quality graph to Figure 5.10. Numerically, the 2D mapping results of DA-exp95 are 30.2% and 28.3% better than the SMACOF on average with $\varepsilon = 10^{-5}$ and $\varepsilon = 10^{-6}$ stop conditions, correspondingly. For the 3D mapping case, SMACOF shows 19.7% worse than DA-exp95 when $\varepsilon = 10^{-5}$, and 2.6% worse than DA-exp95 when $\varepsilon = 10^{-6}$. As usual, DA-SMACOF shows consistent mapping quality by a little variation of the STRESS values for all test cases. The standard deviation of STRESS values of 3D mappings by DA-SMACOF is as small as about 10^{-5} or 10^{-6}.

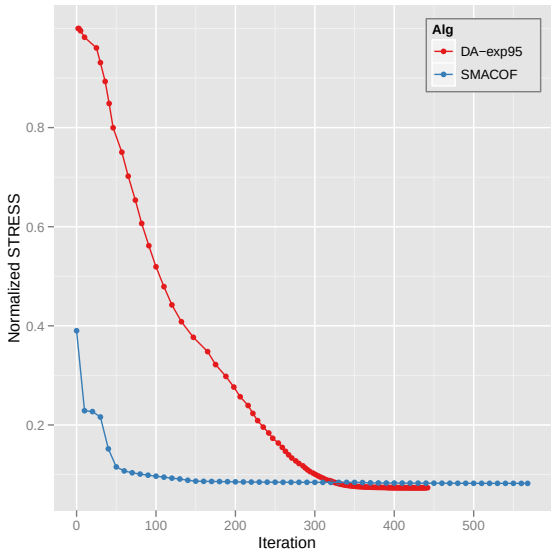
In terms of the linear cooling scheme and the hybrid stop condition approach experiments, Figure 5.12 shows the same tendency to Figure 5.10. Also, there are some unique features of Figure 5.12. The one thing is that the best mapping of all experimental runnings in 2D space is generated by the DA-exp95 with $\varepsilon = 10^{-5}$ stop condition, and the difference with other mappings is perceptible in Figure 5.12-(a). This result

proves that it is possible to find a better mapping by the DA-SMACOF algorithm with a less constrained stop condition (here, $\varepsilon = 10^{-5}$, instead of $\varepsilon = 10^{-6}$). Another unique feature in Figure 5.12, compared to Figure 5.10, is that one of the SMACOF runs in 3D space with the $\varepsilon = 10^{-5}$ stop condition quite prematurely stopped and resulted in generating a poor configuration as shown in Figure 5.12-(b).

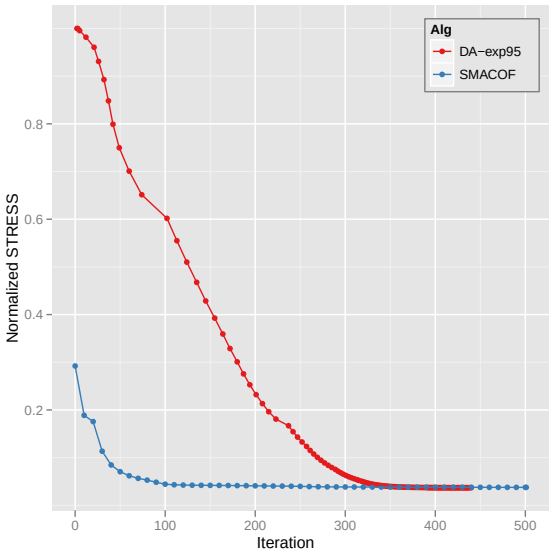
5.5.5 Comparison of the STRESS progress

In addition to the normalized STRESS value comparison between the proposed DA-SMACOF and SMACOF, in order to investigate how to avoid local optima by DA-SMACOF, we would like to look into the STRESS value progress of SMACOF and DA-exp95 with the $\varepsilon = 10^{-6}$ stop condition in the 2D and 3D mapping results. Figure 5.13 and Figure 5.14 illustrate the transition of the STRESS values of each experiment of SMACOF and DA-exp95 algorithms with **MC30000** and **RNA50k**, correspondingly. As shown in Figure 5.13 and Figure 5.14, the DA-SMACOF algorithm starts from a high STRESS value (almost 1.0) and tends to decrease its STRESS value slowly at first, and then experience a faster decrease in the intermediate stage, and converge to a solution with a less STRESS value than the SMACOF, in a reversed 'S' shaped transition for both (a) and (b) subfigures of Figure 5.13 and Figure 5.14. On the other hand, the SMACOF algorithm starts with a smaller initial STRESS value (smaller than 0.5) than the DA-SMACOF but shows a steep descent of the STRESS value progress in the beginning and long tails after the short steep decrease in both 2D and 3D mapping cases in Figure 5.13 and Figure 5.14.

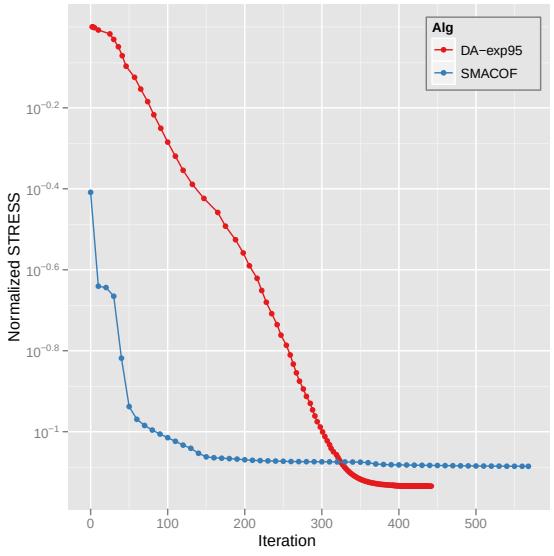
(c) and (d) subgraphs of Figure 5.13 and Figure 5.14 took log-scale on the axis of the normalized STRESS value from (a) and (b) subfigures of Figure 5.13 and Figure 5.14, correspondingly. By log-scaling on the y-axis, we can look into the details of the almost flat regions at high iterations in (a) and (b) subfigures of Figure 5.13 and Figure 5.14. Interestingly, the log-scaling helps to differentiate the DA-SMACOF results in both the 2D and 3D mappings, while the results of the SMACOF algorithm still show some flat regions by even log-scaled graphs, as shown in Figure 5.13 and Figure 5.14.



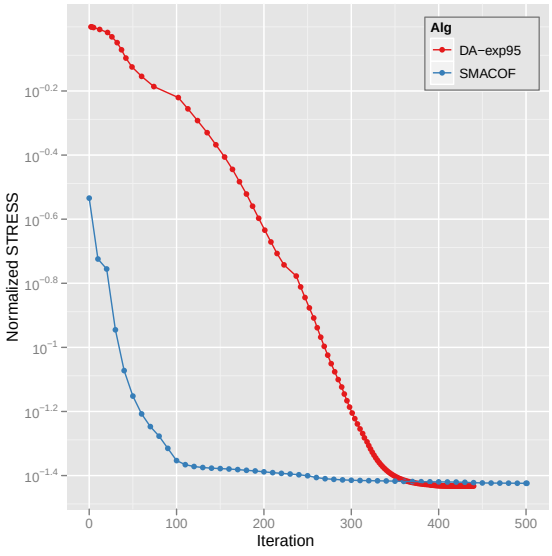
(a) MC30000 2D STRESS Movement



(b) MC30000 3D STRESS Movement



(c) MC30000 2D STRESS Movement - logscale



(d) MC30000 3D STRESS Movement - logscale

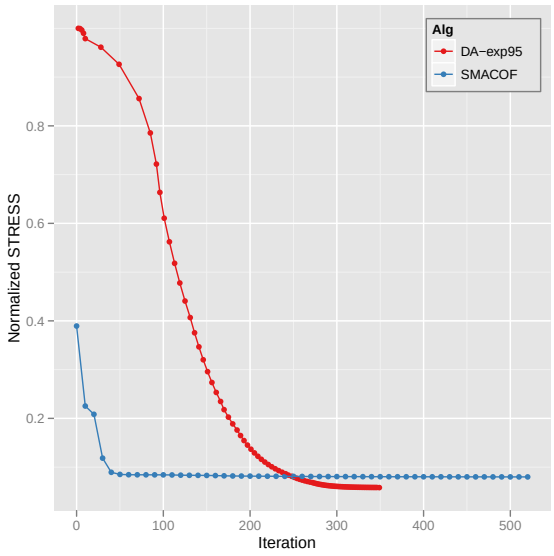
Figure 5.13: The normalized STRESS progress comparison of **MC 30k** data in 2D and 3D space by SMACOF and DA-exp95. The x-axis is the cummulated iteration number of SMACOF and DA-SMACOF algorithms.

Note that the DA-exp95 stops at a lesser number of iterations for both the 2D and 3D mappings than the SMACOF algorithm, and it affects the running time as shown in Figure 5.15-(a) and Figure 5.16. We can interpret the steep improvement of mapping quality by SMACOF in the beginning as being powerful but it results in being trapped in local optima easily, and the gradual improvement by DA-SMACOF based on the change of computational temperature gives an effect in which it can avoid local optima and be efficient for large data sets.

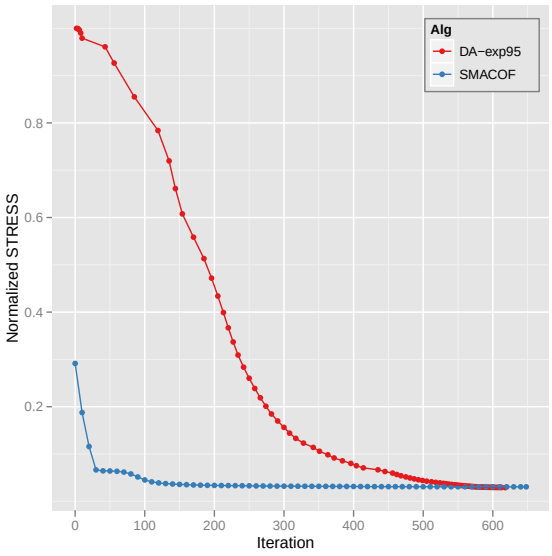
5.5.6 Running Time Analysis of Large Data Sets

I described the runtime analysis of the DA-SMACOF compared to other algorithms (SMACOF and MDS-DistSmooth) with relatively small data sets in Section 5.4.5. In Section 5.4.5, the DA-SMACOF takes awhile longer than the SMACOF but also demonstrates a shorter time than the MDS-DistSmooth algorithm. However, the DA-SMACOF takes a comparable and even shorter time than does the SMACOF with larger data sets in the 2D and 3D mappings, interestingly enough. Since the DA-SMACOF and SMACOF algorithms require too much computing power and main memory, parallelization is essential for testing with large data points. Therefore, we tested DA-SMACOF and SMACOF with the MPI version of these algorithms [16] for testing with larger data sets. The details of the runtime analysis for each large data set will be represented in this section. In Figure 5.15 and Figure 5.16, the bar graph shows the average runtime of 10 runs of each experimental set up, and the error-bar on top of each bar graph represents the minimum and maximum running time of the corresponding experiments.

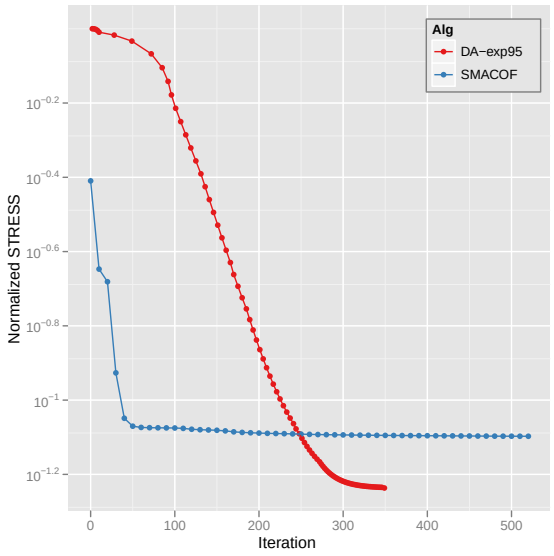
First, for the **metagenomics** data set with 30,000 points, I tested this set with 128 way parallelism by using the MPI version of the SMACOF and DA-SMACOF [16]. The running time results of DA-SMACOF and SMACOF algorithms with the metagenomics data is shown in Figure 5.15-(a). In detail, the DA-SMACOF takes only 1.36 and 1.12 times longer than the SMACOF in average for the 2D mapping results, and several SMACOF runs actually take longer than the DA-SMACOF running times. In contrast to the running time



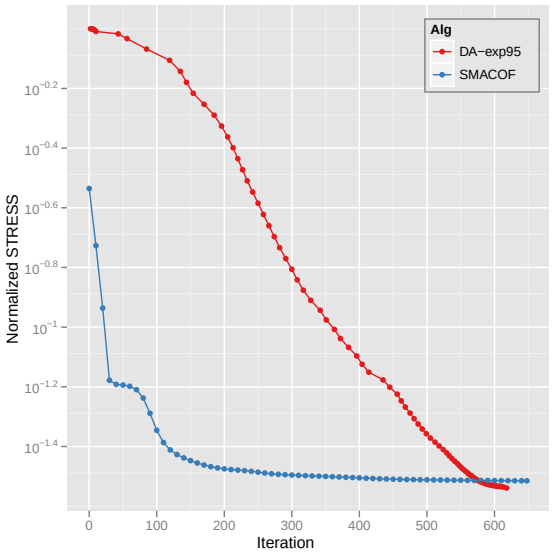
(a) 16sRNA 50k 2D STRESS Movement



(b) 16sRNA 50k 3D STRESS Movement

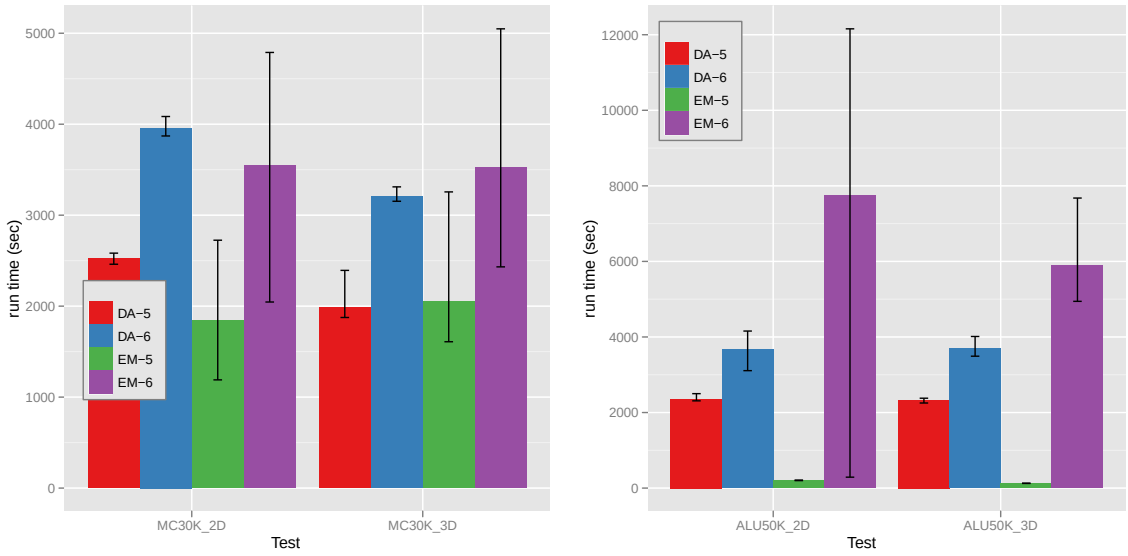


(c) 16sRNA 50k 2D STRESS Movement - logscale



(d) 16sRNA 50k 3D STRESS Movement - logscale

Figure 5.14: The normalized STRESS progress comparison of **RNA50k** data in 2D and 3D space by SMACOF and DA-exp95. The x-axis is the cummulated iteration number of SMACOF and DA-SMACOF algorithms.



(a) MC30000 Data Runtime

(b) ALU50058 Data Runtime

Figure 5.15: The average running time comparison between SMACOF and DA-SMACOF (DA-exp95) for 2D and 3D mappings with MC30000 and ALU50058 data sets. The error bar represents the minimum and maximum running time. **EM-5/EM-6** represents SMACOF with $10^{-5}/10^{-6}$ threshold, and **DA-5/DA-6** represents the runtime results of DA-SMACOF, correspondingly, in the same way.

results of smaller data sets in Section 5.4.5, the DA-SMACOF takes 0.97 and 0.91 times longer than the SMACOF, which means it is faster than the SMACOF, in average for 3D mappings. The DA-SMACOF still shows high consistency in running time as well as STRESS values, but the SMACOF shows high variation of running time in Figure 5.15-(a).

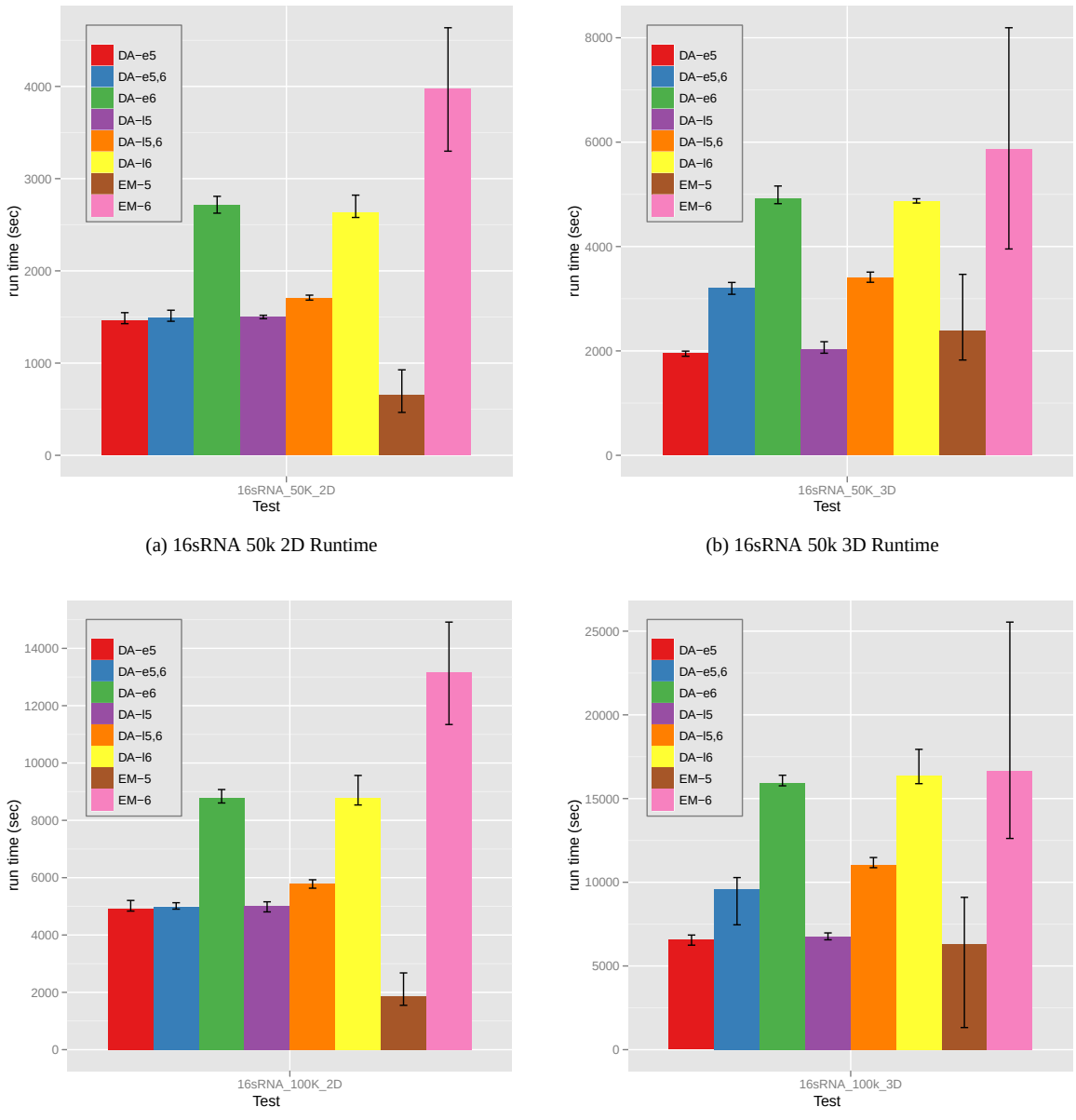
Figure 5.15-(b) is the average running time of each experiment with the **ALU50058** data set. As we mentioned in Section 5.5.2, SMACOF with $\varepsilon = 10^{-5}$ cases stopped prematurely and it represents a very small amount of running time. Since the SMACOF stops with too small a number of iteration with the $\varepsilon = 10^{-5}$ stopping condtion, I would like to analyze the runtime of both algorithms with the $\varepsilon = 10^{-6}$ stopping condition. As shown in Figure 5.15-(b), DA-SMACOF with 10^{-6} stop condition shows much better running time performance than does the SMACOF algorithm with the $\varepsilon = 10^{-6}$ stopping condition in both 2D and 3D mapping results. The DA-SMACOF takes only 0.47 times longer than the SMACOF in 2D mappings,

which means the DA-SMACOF is more than twice as fast as the SMACOF. For the 3D mapping with the ALU50058 data set, the DA-SMACOF takes around 0.63 times longer (a.k.a. 1.60 times faster) than the SMACOF. Note that the SMACOF takes a lot of variation with running time in 2D mapping with $\varepsilon = 10^{-6}$ stop condition.

Figure 5.16 illustrates the running time analysis of the DA-SMACOF and SMACOF algorithm with **16s RNA** data sets, i.e. **RNA50k** and **RNA100k**. For 16s RNA data sets, I experimented with a linear cooling scheme and a hybrid stop condition approach as well as the usual DA-exp95 case. Figure 5.16-(a) and (b) show the running time results with the RNA50k data set in 2D and 3D space, correspondingly, and Figure 5.16-(c) and (d) depict the running time result with the RNA100k data set in 2D and 3D space, as well. As shown in Figure 5.16, Figure 5.16-(a) shows a very similar graph to Figure 5.16-(c), and Figure 5.16-(b) and (d) are similar to each other.

First, I would like to analyze a runtime comparison between DA-SMACOF and SMACOF. I will use same DA-exp95 case to compare with SMACOF to exhibit the same comparison that I have previously conducted with other data sets. As described in Figure 5.16-(a) and (c), for the 2D mapping results, the SMACOF with the $\varepsilon = 10^{-5}$ stopping condition stopped faster than the DA-exp95 with the same stop condition. However, if we compare the running time of DA-exp95 and SMACOF with $\varepsilon = 10^{-6}$ stop condition in 2D mappings, the DA-SMACOF takes around 0.68 and 0.67 times longer than the SMACOF algorithm. In other words, the DA-SMACOF is about 1.47 and 1.50 times faster than the SMACOF. Note that the mapping quality of the SMACOF with the $\varepsilon = 10^{-5}$ stopping condition for the 2D space is much worse than that of the DA-exp95, as explained in Section 5.5.3.

For 3D mappings, even the SMACOF with the $\varepsilon = 10^{-5}$ stopping condition is slower than or comparable to the DA-SMACOF with $\varepsilon = 10^{-5}$, and the SMACOF takes longer than the DA-SMACOF with the $\varepsilon = 10^{-6}$ stopping condition as shown in Figure 5.16-(b) and (d). Numerically, the DA-SMACOF takes around 0.82 times longer (actually, 1.22 times faster) than the SMACOF with the RNA50k data set and takes only 1.04



(a) 16sRNA 50k 2D Runtime

(b) 16sRNA 50k 3D Runtime

(c) 16sRNA 100k 2D Runtime

(d) 16sRNA 100k 3D Runtime

Figure 5.16: The average running time comparison between SMACOF, DA-SMACOF (DA-exp95), and DA-SMACOF with linear cooling (DA-lin100) for 2D and 3D mappings with 16s RNA data sets. The error bar represents the minimum and maximum running time. **EM-5/EM-6** represents SMACOF with $10^{-5}/10^{-6}$ threshold, and **DA-e5/DA-e5,6/DA-6** and **DA-l5/DA-l5,6/DA-l6** represents the runtime results of DA-exp95 and DA-lin100, correspondingly, with 10^{-5}/hybrid(10^{-5} and 10^{-6})/10^{-6}.

times longer than the SMACOF with the RNA100k data set, when $\varepsilon = 10^{-5}$. Also, the DA-SMACOF with $\varepsilon = 10^{-6}$ stop condition is about 1.19 times and 1.04 times faster than the SMACOF with same stopping condition for RNA50k and RNA100k data sets, correspondingly.

If we compare the running times between two different computational temperature cooling methods for DA-SMACOF, we can find that the **DA-exp95** and **DA-lin100** are compatible with the $\varepsilon = 10^{-5}$ stopping condition and the DA-lin100 shows better than or comparable to performance to the DA-exp95 with the $\varepsilon = 10^{-6}$ stopping condition. It is interesting that the DA-lin100 shows better than or comparable to performance with the DA-exp95 in terms of the running time, based on the fact that the tested linear cooling mechanism takes more cooling steps than the exponential cooling scheme as shown in Figure 5.1. From the above facts, we can deduce that the linear cooling scheme is a little bit more efficient than the exponential cooling scheme in terms of running time.

The hybrid stop condition approach, which uses two different iteration stop conditions for the annealing period ($T > 0$) and the final step ($T = 0$), shows interesting attributes with respect to the running time analysis in Figure 5.16. For 2D mapping cases as in Figure 5.16-(a) and (c), the hybrid approach shows a similar running time with a corresponding $\varepsilon = 10^{-5}$ stop condition. On the other hand, the DA-SMACOF with a hybrid stop condition scheme takes a longer time than the DA-SMACOF with $\varepsilon = 10^{-5}$ but a shorter time than the DA-SMACOF with $\varepsilon = 10^{-6}$ for 3D mapping cases. Since the mapping quality of the DA-SMACOF hybrid stop condition scheme is similar to the DA-SMACOF with $\varepsilon = 10^{-5}$ in 2D mappings and similar to the DA-SMACOF with the $\varepsilon = 10^{-6}$ in the 3D mappings as illustrated in Figure 5.10 and Figure 5.12, we can think of the running time of the hybrid stop condition approach shown in Figure 5.16 as a reasonable result. Since the hybrid stop condition scheme for the DA-SMACOF shows a similar mapping quality to the DA-SMACOF with $\varepsilon = 10^{-6}$ for 3D mappings, if we compare the runtime of the hybrid stop condition scheme with the corresponding DA-SMACOF with $\varepsilon = 10^{-6}$ experiments in 3D mappings, the hybrid stop condition achieves a 1.53 and 1.43 times faster running time than DA-SMACOF with $\varepsilon = 10^{-6}$ for the RNA50k data,

and 1.66 and 1.48 times faster than DA-SMACOF with $\varepsilon = 10^{-6}$ for the RNA100k data with respect to the exponential and linear cooling schemes, correspondingly. Furthermore, the hybrid stop condition approach for DA-exp95 case is about 1.82 and 1.73 times faster than the SMACOF with $\varepsilon = 10^{-6}$ for RNA50k and RNA100k data, respectively.

In short, the DA-SMACOF generally outperforms the SMACOF, not only in mapping quality but also in running time, with large data sets. Based on the runtime analysis of the DA-SMACOF with a large data set in this section, we can consider the proposed DA approach for MDS as not just an effective optimization method for avoiding local optima but also an efficient optimization method for large data sets.

5.6 Summary

In this chapter, we have proposed an MDS solution with the deterministic annealing (DA) approach, which utilizes the SMACOF algorithm in each cooling step. The proposed DA approach outperforms the SMACOF and MDS-DistSmooth algorithms with respect to the mapping qualities with several different real data sets. Furthermore, the DA-SMACOF exhibits the high consistency due to less sensitivity to the initial configurations, in contrast to the SMACOF and MDS-DistSmooth approaches, which show high sensitivity to both the initial configurations and the stopping condition.

With the benefit of the DA method to avoid local optima, the proposed DA approach uses slightly longer or comparable running times to the SMACOF, and shorter running times than the MDS-DistSmooth approach, for small data sets. Moreover, the proposed DA approach is actually faster than the SMACOF algorithm with larger data sets, and applying the hybrid stop condition scheme to the DA-SMACOF algorithm can reduce the running time of the DA algorithm even more.

We have also investigated different computational temperature cooling parameters in an exponential cooling scheme and it turns out that this approach shows some deviation of mapping results when we use a faster

cooling parameter than necessary (like the DA-exp90 case in this chapter). But the DA-exp90 shows still better than or comparable performance compared with the compared algorithms in our experiments. Also, the DA-exp95 results are very similar to or even better than the DA-exp99 results, although the DA-exp95 takes a shorter time than the DA-exp99 case, so that we might think $\alpha = 0.95$ could be a generally reasonable cooling parameter in an exponential scheme. In addition to the exponential cooling scheme, we also tested with linear cooling scheme and compared both cooling scheme in terms of mapping quality as well as run-times. The linear cooling scheme shows slightly better than or comparable to performance of the exponential cooling scheme in both quality and efficiency criteria.

For the scalability test of the ability of avoiding local optima by the DA-SMACOF, we experimented with various sizes of data sets from 150 to 100,000 points, and compared them with the SMACOF (and MDS-DistSmooth for smaller data sets) applied to the same data sets. In Section 5.4 and Section 5.5, the experimental results support the local optima avoidance attribute of the proposed DA-SMACOF algorithm, not only for small data sets but also for large data sets.

6

Conclusions and Future Works

6.1 Summary of Work

In this dissertation, we have worked on a dimension reduction method called multidimensional scaling (MDS). The main purpose of my dissertation work is to scale up the mapping capacity of the MDS algorithm for dealing with a large amount of data, and to improve the mapping quality of MDS results via avoiding local optima. First, we applied parallelism to a well-known MDS algorithm called SMACOF [20, 21] in order to increase the computational capacity of the MDS algorithm through utilizing more computing resources on cluster systems for dealing with large amounts of data in Chapter 3. We also investigated the parallel performance and the scalability of our parallel implementation in Chapter 3 as well. In addition to applying parallelism in order to utilize more computation resources, we proposed an interpolation algorithm which reduces computational complexity and memory requirement in Chapter 4. The interpolation approach divides the given data in two sets, i.e. sample data and out-of-sample data. Then, we generate a configuration of the sample data with the full MDS algorithm, and we find a mapping of each out-of-sample point by means of interpolation based on the mappings of nearest neighbors of the interpolated point among the sample data. We can proceed with including millions of points by using the proposed interpolation method as shown

in Chapter 4. For the purpose of achieving better mapping quality, we applied an optimization method named deterministic annealing (DA) [62, 63] to the MDS problem in Chapter 5. We tested the proposed DA-SMACOF with various data sets with a variety of the sizes compared to other MDS algorithms, and the DA-SMACOF produced better quality mappings than the other comparable MDS algorithms.

6.2 Conclusions

Large-scale data analysis is a prominent research area due to the data explosion which has occurred in almost every domain. Huge amounts of data are generated, not only from the scientific and technical area but also from personal life activities, such as digital pictures, video clips, postings on a personal blog system or social network media, and so on. The dimension reduction algorithms aim to generate low-dimensional human-perceivable configurations which are very useful for investigating high-dimensional data sets. Among the many dimension reduction algorithms, we focus on the multidimensional scaling (MDS) algorithm in this dissertation, due to its robustness and high applicability.

We have worked on several ways to improve a well-known MDS algorithm, called SMACOF [20, 21], with respect to not only computing capability but also mapping quality. To increase the possible number of points generated in a new configuration in a target dimension, we have worked on the parallelization of SMACOF algorithm. The parallelization enables the SMACOF algorithm to deal with hundreds of thousands of points via distributed multicore cluster systems, such as 32 nodes with 768 cores. Although the parallel SMACOF implementation provides much more computing power, it cannot affordably configure millions of points since the computational complexity and memory requirement of the SMACOF algorithm is still $\mathcal{O}(N^2)$. The proposed majorizing interpolation MDS (MI-MDS) makes possible the strategy generating a mapping of millions of points with a trade-off between the computing capacity and the mapping quality. We have worked on the improvement of mapping quality by avoiding local optima as well as on the increase of

the computing capacity of the MDS solution. Below we have summarized what we have worked out in each chapter of this dissertation.

6.2.1 High-Performance Distributed Parallel Multidimensional Scaling (MDS)

In order to configure mappings with large amounts of data through the MDS algorithm, we need to utilize distributed computing systems due to the requirement of a large amount of computing power and memory. Thus, we parallelized a well-known MDS solution named SMACOF [20, 21] via MPI [27, 73]. The MPI standard has been supported in most programming languages, such as C, C++, Java, and C#, and MPI.NET [32], which we used in this thesis, provides MPI implementation for the C# language.

As we mentioned in Chapter 3, we implemented the scalable and high-performance parallel SMACOF algorithm which shows high efficiency and scalability. In Chapter 3, we demonstrated how the importance of the data decomposition structure can influence message passing routines and overhead as well as a cache-line effect. We measured both overall elapsed running time and sub-routine runtimes with respect to various data decomposition structures, and found that the row-based decomposition worsens the cache reusability for updating the distance matrix with larger data sets. The worse cache reusability results in performance degradation of row-based decomposition with larger data set. On the other hand, column-based decomposition also increases message passing overhead for updating another participating matrix ($B(X)$) of the SMACOF algorithm. Based on both experimental results, we conclude that the *balanced* data decomposition is generally better than the *skewed* data decomposition for parallel SMACOF algorithm.

The efficiency and scalability analyses are important criteria for the evaluation of parallel implementation. Although the efficiency is decreased, the decreased efficiency is still good enough for a certain degree of parallelism. The main reason for the efficiency degradation of the parallel SMACOF implementation, as the number of parallel units being increased, is that the ratio of parallel overhead portion is increased but the pure parallel computation time is decreased as the number of parallel units is increased. Figure 3.8 and Table 3.3

supported two important aspects of the proposed parallel SMACOF: (1) the fact that it achieves good load-balance; and (2) that the decreased efficiency of the parallel SMACOF with more parallel units is due to the inevitable message passing overhead for the parallel implementation.

In short, the major contribution of the parallel SMACOF implementation is that it can afford to utilize cluster systems, which provide hundreds or even thousands times more computing power and memory resources than a single node machine, for running the MDS algorithm with a much larger number of data points, such as hundreds of thousands of points that would be impossible to run in a single node machine by sequential computing. The proposed parallel SMACOF algorithm shows relatively high efficiency and scalability by paying some inevitable overhead.

6.2.2 Interpolation approach for MDS

By using the parallel SMACOF algorithm on a cluster system, we can configure hundreds of thousands of points in a target dimensional space. However, there are many interesting problems with more than millions of points, in that it is impractical to use only the parallel SMACOF implementation to configure mappings for millions of points in a target dimension due to the resource requirements of $\mathcal{O}(N^2)$ concerning computation and memory. In order to raise the affordable number of points of the MDS algorithm at least up to millions of points, we developed an interpolation approach for the MDS problem. The interpolation method for a given data follows a two-fold operations: (1) we select samples from the given full data and generate mappings of the *sample* data; (2) we then interpolate each *out-of-sample* data with respect to the mapping positions of its nearest neighbors among sampled data. By applying the interpolation approach, we reduce the computational complexity of the MDS algorithm from $\mathcal{O}(N^2)$ to $\mathcal{O}(nN)$ where N is the total data size and n is the sample data size. We applied the iterative majorization method to interpolate a points with respect to the mapping positions of its nearest neighbors. The mathematical equations for the proposed iterative majorizing interpolation method are provided in Chapter 4. In Chapter 4, we also illustrated how to parallelize the

interpolation algorithms.

In Chapter 4, we discussed: (1) the comparative experimental analysis between the proposed majorizing interpolation MDS (hereafter MI-MDS) and MDS only in terms of mapping quality and execution time; and (2) the parallel performance of MI-MDS algorithm. For mapping quality comparison purposes, the normalized STRESS value of interpolation approach, which combines with MDS running with the sample data and MI-MDS with the out-of-sample data, is compared to the MDS only running. For the Pubchem 100k data set, the normalized STRESS value of the interpolation approach with a 50k sample data is only around 0.0038 bigger than the results of the MDS only. Note that the interpolation approach is 3.3 times faster than the MDS only approach, and that the MI-MDS takes only around 19 seconds to generate mappings of 50k out-of-sample data. We also experimented with larger out-of-sample data size, such as 1 million, 2 millions, and 4 millions points (hereafter 1M, 2M, and 4M, correspondingly). Since we cannot run parallel MDS with more than 200,000 data points on Cluster-II in Table 4.1 due to the out of memory exception, there is no way to compare the mapping quality of the interpolation approach result with MDS only results. However, we can evaluate the mapping quality of the interpolation results with larger data sizes (like millions) by comparing them to the normalized STRESS values of the MDS result of sampled data. As we expected, the approximation feature of MI-MDS degraded the mapping quality a little, but the mapping qualities of MI-MDS results with 1M, 2M, and 4M data points are similar to one another as shown in Figure 4.9, interestingly enough.

In addition to the mapping quality analysis of the MI-MDS, we also investigated the parallel performance of the hybrid parallel MI-MDS algorithm in Chapter 4. The independence between the interpolated points is a very nice feature to parallelize the MI-MDS algorithm. The parallel MI-MDS algorithm is a pleasingly parallel application due to the independence between each interpolated points. Thus, the parallel MI-MDS implementation achieves high efficiency in parallel as shown in Section 4.4.3.

In short, the proposed interpolation algorithm (MI-MDS) succeeds to generate a configuration of millions

of points in a target dimension in moderate running time. The mapping quality of the outputs from the interpolation approach is comparable to the mapping quality of the MDS only results, which takes a much longer execution time than the interpolation approach. We parallelized the proposed MI-MDS algorithm to deal with millions of out-of-sample data with high efficiency due to the independent properties of the interpolated points.

6.2.3 Improvement in Mapping Quality

By working on the parallelization of MDS algorithm and development of interpolation algorithm, we have contributed in quantitative scope to the MDS algorithm for the purpose of dealing with large-scale input data. In addition to this quantitative contribution, we have also worked on a method to avoid the local-optima problem of the MDS algorithm which results in the contribution of mapping quality improvement. For the sake of overcoming the local-optima problem on MDS, we applied a well-known optimization method, called the Deterministic Annealing (DA) [62, 63], to the MDS problem. In Chapter 5, we proposed an MDS solution with a DA optimization approach. The SMACOF [20, 21] algorithm is used to generate an approximated mapping for the given data at each cooling step (T) in the proposed DA MDS solution. (Hereafter, we call the proposed DA MDS solution the *DA-SMACOF*.)

In order to experiment with the local-optima avoidance property, we compared the DA-SMACOF algorithm with other MDS algorithms, i.e. the SMACOF and MDS-DistSmooth (MDS by distance smoothing [35]), with various real data sets. The experimental results are shown in Section 5.4 and Section 5.5. The experimental analysis illustrates that the DA-SMACOF outperformed other comparable algorithms, the SMACOF and MDS-DistSmooth, in terms of mapping quality. Also, a high consistency of mapping results is exhibited among the DA-SMACOF results and it confirms that the DA-SMACOF's decreased sensitivity to intial configurations. On the other hand, the SMACOF and MDS-DistSmooth are highly sensitive to the initial configurations and the stopping condition parameter. In order to see the difference of the optimization

activity between the SMACOF and DA-SMACOF, we investigated the trace of the STRESS value by each algorithms. As shown in Figure 5.14, the trace of the STRESS value for the DA-SMACOF shows a reversed 'S' shaped progress, which starts from a very high STRESS value and slow improvement initially. On the other hand, the SMACOF case shows 'L' shaped progress, which illustrates faster improvement from the beginning with a much lower starting STRESS value followed by long tailed small improvement. It is a natural interpretation for the STRESS value trace results that the steepest improvement by the SMACOF results in being trapped in local optima but the gradual progress by the DA-SMACOF related to the computational temperature demonstrates its avoidance of local-optima.

In comparison for the running time, the proposed DA-SMACOF takes comparable to or slightly longer than the SMACOF algorithm but it uses a much shorter running time than the MDS-DistSmooth with better mapping results, for the small data sets in Section 5.4. Furthermore, the DA-SMACOF is even faster than the SMACOF algorithm with larger data sets as in Section 5.5, and it produced better quality mappings than the SMACOF. We can reduce the DA-SMACOF running time even more by applying the hybrid stop condition scheme to the DA-SMACOF, as shown in Figure 5.16.

In addition to a comparative analysis conducted between DA-SMACOF and other well-known MDS algorithms, we have experimented with various parameters of the DA-SMACOF as well, such as temperature cooling parameters and the stop condition parameter. First, we have worked on different temperature cooling parameters (i.e. α = 0.90, 0.95, and 0.99 denoted *DA-exp90, DA-exp95, and DA-exp99*, correspondingly) in an exponential computational temperature cooling scheme. The case of $\alpha = 0.90$ is a kind of faster cooling parameter than appropriate, and the outputs of DA-exp90 generated somewhat variated STRESS value distributions, in contrast to the DA-exp95 and DA-exp99 cases, but the DA-exp90 shows still better than or comparable to performance to the SMACOF and MDS-DistSmooth algorithms in Section 5.4. Furthermore, the DA-exp95 performs similar to or even better than the DA-exp99, and it takes less time than the DA-exp99 case. Based on the experiment of the temperature cooling parameter in the exponential cooling scheme, we

conclude that $\alpha = 0.95$ in an exponential cooling mechanism could generally be a reasonable choice. Second, we also compared the exponential and linear cooling schemes in terms of mapping quality and execution time. The linear cooling scheme shows a little bit better than or comparable to performance to the exponential cooling scheme not only in mapping quality but also in running time.

We experimented the proposed DA-SMACOF algorithm with eight different real data sets and compared the mapping quality with that of other algorithms. The size of data sets varies from 150 to 100,000 points. For the all test cases, the DA-SMACOF shows better STRESS value than the compared algorithms without regard to the size of data. It shows that the proposed DA-SMACOF avoids local optima, as the algorithm intended, not only for small data sets but also for large amounts of data sets. The detailed experimental results of those data sets are showin in Section 5.4 and Section 5.5.

6.3 Future Works

In this section, I would like to mention about some possible future works related to the proposed methodologies in this thesis. Those future research issues will improve the features of the proposed algorithms in this thesis.

6.3.1 Hybrid Parallel MDS

In [28, 57], we investigated the overhead of a pure MPI and a hybrid (MPI-Threading) model with multicore cluster systems. In [28], the pure MPI outperforms the hybrid model for the application with a relatively fast message passing synchronization overhead. However, for the case of the high MPI synchronization time, the hybrid model outperforms the pure MPI model with high parallelism. Since the MPI overhead increases as the number of processes is increased in Figure 3.9, it is worth to investigate hybrid parallel model of SMACOF.

6.3.2 Hierarchical Interpolation Approach

In Chapter 4, we discussed an interpolation approach which reduces the time complexity of the MDS algorithm from $\mathcal{O}(N^2)$ to $\mathcal{O}(nM)$, where n is the sample size and M is the out-of-sample size ($M = N - n$). MI-MDS provides us with a very high capability of configuring large amounts of data points in a target dimension via a modest amount of execution time. Although the time complexity of the MI-MDS is much better than that of the normal MDS algorithm, and it shows efficient running times compared to running times of the parallel SMACOF as described in Chapter 4, we can even reduce the time complexity to $\mathcal{O}(M \cdot log(n))$ by using a hierarchical approach, such as the Barnes-Hut tree [8] (hereafter *BH-tree*). The BH-tree technique divides the given space by a tree-structured subspace of cubic cells, and the division occurs recursively in each cubic cells. For instance, the volume of 2D and 3D space is divided by a *quadtree* and an *octree* structure, correspondingly. The BH-tree technique is originally proposed to advance the time complexity of the *N-body* simulation. We can reduce in log-scale the number of points which needs to be considered as possible neighbors among the sampled data of each interpolated point by applying the BH-tree technique. Currently, a colleague of our research group has been working on this hierarchical interpolation approach for MDS problem.

6.3.3 Future Works for DA-SMACOF

It will be interesting to integrate DA-SMACOF ideas with the interpolation technology described in [7] to give a robust approach to dimension reduction of large datasets that scales like $\mathcal{O}(nN)$ rather $\mathcal{O}(N^2)$ of general MDS methods. The adaptive cooling scheme for computational temperature of DA-SMACOF is another interesting topic for further research. The adaptive cooling scheme may result in more efficient running on DA-SMACOF by changing computational temperature based on the progress of the algorithm. Although we have worked on mainly the DA-SMACOF with a uniform weighted case (unweighed case, $w_{ij} = constant$) in Chapter 5, DA-SMACOF algorithm for weighted case, such as Sammon's mapping ($w_{ij} = 1/\delta_{ij}$), could be

also useful under certain environments. Therefore, it is an interesting research topic to extend the proposed DA-SMACOF to a generalized DA MDS algorithm which can deal with general weighted cases as well as unweighted case.

6.4 Contributions

The followings are the envisioned contributions of this dissertation in Section 1.5:

- **[Parallelization]** Efficient parallel implementation via Message Passing Interface (MPI) in order to scale up the computing capability of a well-known MDS algorithm (SMACOF) by utilizing cluster systems.
- **[Reducing Complexity]** Development of an innovative algorithm, which reduces the computational complexity and memory requirement of MDS algorithms and produced acceptable mapping results, for the purpose of scaling the MDS algorithm's capacity up to millions of points which is usually intractable to generate a mapping via normal MDS algorithms.
- **[Local Optima Avoidance]** Providing an MDS algorithm which could figure out the local optima avoidance problem in a deterministic way so that it generates better quality mapping in a reasonable amount of time.

In this section, we summarize how to achieve the envisioned contributions in this dissertation that we mentioned in Section 1.5. The methods of how we achieved the proposed contributions in this dissertation are shown below:

- **[Parallelization]** We parallelized the SMACOF algorithm [20, 21] via MPI.NET [32] and C# language which is a managed code. The implemented parallel SMACOF algorithm shows the achievement of

good load-balance. Due to the inevitable messages on each iteration of SMACOF algorithm, the parallel efficiency is lowered as the number of parallel units is increased. However, the parallel efficiency is still good enough with an appropriate number of parallel units with respect to the data size. The computing capability of the parallel SMACOF is scaled up to hundreds of thousands of points on the multicore cluster systems that we have.

- **[Reducing Complexity]** We designed an interpolation algorithm which divides the given data into two sets as *sample* data and *out-of-sample* data. The interpolation algorithm consists of two-steps: (1) mapping sample data by full MDS running; and (2) intepolate the out-of-sample data based on the mappings of sample data generated by (1) step. We designed an interpolation algorithm which reduces the computation complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(nM)$, where n is the size of sample data and $M = N - n$. Also, the proposed interpolation algorithm reduces the memory requirement for generating a mapping position for each point. Due to the reduced computational complexity and memory requirement, the proposed interpolation algorithm actually scales the mapping capacity up to millions of points not only with a much faster running time but also with an acceptable mapping quality. As shown in Chapter 4, the intepolation approach enables us to configure up to 4-million points in 3D space.

- **[Local Optima Avoidance]** In addition to the contributions of scalability of the MDS algorithm, we also solved the avoidance of the local optima issue which is a well-known problem of EM-like optimization methods. We have applied deterministic annealing (DA) [62, 63] optimization to the MDS algorithm to avoid local optima via simplifying the expected energy function by approximation based on computational temperature (T). The unique attribute of the DA optimization method is that it is a deterministic approach, which does not rely on non-deterministic random movement, in contrary to other global optimization methods, such as simulated annealing (SA) and genetic algorithm (GA), so that it does not suffer from long running times. Since we utilized the SMACOF algorithm to find a mapping of the given data at each temperature (T), we could also parallelize the proposed DA-SMACOF

algorithm by using the parallel SMACOF implementation [16] which is proposed in Chapter 3. The proposed DA-SMACOF algorithm depicted the local optima avoiding property by high-consistency of mapping quality and less-sensitivity to initial configuration. We compared the proposed DA-SMACOF algorithm with other algorithms by running with various size of data sets from 150 to 100,000 as in Section 5.4 and Section 5.5. It outperformed other compared MDS algorithms, i.e. SMACOF [20, 21] and MDS-DistSmooth [35], in terms of mapping quality with various data sets, and we could conclude that the better mapping quality of DA-SMACOF results from avoiding local optima for which we designed it.

Bibliography

[1] Boost c++ libraries. http://www.boost.org.

[2] Intel threading building blocks for open source. http://threadingbuildingblocks.org/.

[3] Openmp: The openmp specification for parallel programming. http://openmp.org/wp.

[4] D. Agrafiotis, D. Rassokhin, and V. Lobanov. Multidimensional scaling and visualization of large molecular similarity tables. *Journal of Computational Chemistry*, 22(5):488–500, 2001.

[5] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Berkeley, California, Dec 2006. http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html.

[6] S.-H. 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.

[7] S.-H. Bae, J. Y. Choi, X. Qiu, and G. Fox. Dimension reduction and visualization of large high-dimensional data via interpolation. In *Proceedings of the ACM International Symposium on High Performance Distributed Computing (HPDC) 2010*, Chicago, Illinois, June 2010.

[8] J. Barnes and P. Hut. A hierarchical o(n log n) force-calculation algorithm. *Nature*, 324:446–449, 1986.

[9] M. Batzer and P. Deininger. Alu repeats and human genomic diversity. *Nature Reviews Genetics*, 3(5):370–379, 2002.

[10] 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.

[11] 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.

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

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

[14] M. Brusco. A simulated annealing heuristic for unidimensional and multidimensional (city-block) scaling of symmetric proximity matrices. *Journal of Classification*, 18(1):3–33, 2001.

[15] D. R. Butenhof. *Programming with POSIX threads*. Addison-Wesley Professional, 1997.

[16] 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.

[17] J. Y. Choi, X. Qiu, M. Pierce, and G. Fox. Generative topographic mapping by deterministic annealing. In *Proceedings of the 10th International Conference on Computational Science (ICCS) 2010*, Amsterdam, The Nethelands, June 2010.

[18] G. Chrysanthakopoulos and S. Singh. An asynchronous messaging library for c#. In *Proceedings of the Workshop on Synchronization and Concurrency in Object-Oriented Languages*, pages 89–97, 2005.

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

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

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

[22] 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.

[23] J. Dongarra, D. Gannon, G. Fox, and K. Kennedy. The impact of multicore on computational science software. *CTWatch Quarterly*, 3(1), Feb 2007. `http://www.ctwatch.org/quarterly/archives/february-2007`.

[24] P. Dubey. Recognition, mining and synthesis moves computers to the era of tera. *Technology@Intel Magazine*, 2005.

[25] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[26] M. P. I. Forum. Mpi-2: Extensions to the message passing interface. `http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html`.

[27] M. P. I. Forum. Mpi: A message passing interface standard. `http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html`.

[28] 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.

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

[30] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, 1982.

[31] J. Gower. Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, 53(3-4):325–338, 1966.

[32] D. Gregor and A. Lumsdaine. Design and implementation of a high-performance mpi for c# and the common language infrastructure. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pages 133–142. ACM, 2008.

[33] P. Groenen and P. Franses. Visualizing time-varying correlations across stock markets. *Journal of Empirical Finance*, 7(2):155–172, 2000.

[34] P. Groenen and W. Heiser. The tunneling method for global optimization in multidimensional scaling. *Psychometrika*, 61(3):529–550, 1996.

[35] P. Groenen, W. Heiser, and J. Meulman. Global optimization in least-squares multidimensional scaling by distance smoothing. *Journal of classification*, 16(2):225–254, 1999.

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

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

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

[39] B. Kempf. The boost.threads library. *Dr. Dobb's*, 2002.

[40] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

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

[42] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International joint Conference on artificial intelligence*, volume 14, pages 1137–1145. Morgan Kaufmann, 1995.

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

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

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

[46] H. Lahdesmaki, X. Hao, B. Sun, L. Hu, O. Yli-Harja, I. Shmulevich, and W. Zhang. Distinguishing key biological pathways between primary breast cancers and their lymph node metastases by gene function-based clustering analysis. *International journal of oncology*, 24(6):1589–1596, 2004.

[47] D. Leijen, W. Schulte, and S. Burckhardt. The design of a task parallel library. In *Proceedings of the*

24th ACM SIGPLAN conference on Object oriented programming systems languages and applications, OOPSLA '09, pages 227–242, New York, NY, USA, 2009. ACM.

[48] E. Lessa. Multidimensional analysis of geographic genetic structure. *Systematic Biology*, 39(3):242–252, 1990.

[49] R. Mathar. A hybrid global optimization algorithm for multidimensional scaling. In *Classification and knowledge organization: proceedings of the 20th annual conference of the Gesellschaft für Klassifikation eV University of Freiburg, March 6-8, 1996*, pages 63–71. Springer Verlag, 1997.

[50] R. Mathar and A. Žilinskas. On global optimization in two-dimensional scaling. *Acta Applicandae Mathematicae: An International Survey Journal on Applying Mathematics and Mathematical Applications*, 33(1):109–118, 1993.

[51] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

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

[53] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

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

[55] J. Pješivac-Grbović, T. Angskun, G. Bosilca, G. Fagg, E. Gabriel, and J. Dongarra. Performance analysis of mpi collective operations. *Cluster Computing*, 10(2):127–143, 2007.

[56] J. Qiu and S.-H. Bae. Performance of windows multicore systems on threading and mpi. *Concurrency and Computation: Practice and Experience*, 2011.

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

[58] X. Qiu, G. C. Fox, H. Yuan, S.-H. Bae, G. Chrysanthakopoulos, and H. F. 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.

[59] R. Rabenseifner, G. Hager, and G. Jost. Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes. In *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*, pages 427–436. IEEE, 2009.

[60] J. Reinders. *Intel threading building blocks: outfitting C++ for multi-core processor parallelism*. O'Reilly Media, Inc., 2007.

[61] J. Richter. Concurrent affairs-concurrency and coordination runtime. *MSDN Magazine-Louisville*, pages 117–128, 2006.

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

[63] K. Rose, E. Gurewitz, and G. C. Fox. A deterministic annealing approach to clustering. *Pattern Recognition Letters*, 11(9):589–594, 1990.

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

[65] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[66] H. Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3), 2005. `http://www.gotw.ca/publications/concurrency-ddj.htm`.

[67] 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.

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

[69] W. S. Torgerson. *Theory and methods of scaling.* Wiley, New York, U.S.A., 1958.

[70] 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.

[71] J. Tzeng, H. Lu, and W. Li. Multidimensional scaling for large genomic data sets. *BMC bioinformatics*, 9(1):179, 2008.

[72] J. Vera, W. Heiser, and A. Murillo. Global optimization in any Minkowski metric: a permutation-translation simulated annealing algorithm for multidimensional scaling. *Journal of Classification*, 24(2):277–301, 2007.

[73] D. Walker and J. Dongarra. Mpi: a standard message passing interface. *Supercomputer*, 12:56–68, 1996.

[74] 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.

[75] G. Young and A. Householder. Discussion of a set of points in terms of their mutual distances. *Psychometrika*, 3(1):19–22, 1938.

[76] A. Zilinskas and J. Zilinskas. Parallel genetic algorithm: assessment of performance in multidimensional scaling. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1492–1501, New York, NY, USA, 2007. ACM.

Curriculum Vitae

Name:	Seung-Hee Bae
Date of Birth:	November 13, 1975
Place of Birth:	Seoul, Korea

Education:

February, 2004	Master of Science, Computer Science and Engineering Seoul National University, Seoul, Korea
February, 2002	Bachelor of Engineering, Computer Science and Electronic Engineering Handong Global University, Pohang, Korea

Experience:

January, 2007 - Present	Research Assistant Pervasive Technology Institute, Indiana University Bloomington, Indiana
January, 2006 - December, 2006	Research Assistant School of Informatics, Indiana University Bloomington, Indiana

Honors:

Hyundong Scholarship (March, 1995 - February, 1997)
Handong Global University, Pohang, Korea