ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES

MSc THESIS

Drilon JAHIRI

PYRAMID SELF ORGANIZING MAPS (PSOM): A NEW APPROACH TO ANALYZE LARGE DATA SETS

DEPARTMENT OF COMPUTER ENGINEERING

ADANA, 2015

ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES

PYRAMID SELF ORGANIZING MAPS (PSOM): A NEW APPROACH TO ANALYZE LARGE DATA SETS

Drilon JAHIRI

MSc THESIS

DEPARTMENT OF COMPUTER ENGINEERING

We certify that the thesis titled above was reviewed and approved for the award of degree of the Master of Science by the board of jury on 23/01/2015.

…….....	…….....	…….....
Assoc. Prof. Dr. Mustafa ORAL	Assoc. Prof. Dr. Selma Ayşe ÖZEL	Assoc. Prof. Dr. Oya H. YÜREĞİR
SUPERVISOR	MEMBER	MEMBER

This MSc Thesis is written at the Department of Institute of Natural And Applied Sciences of Çukurova University.
Registration Number:

Prof. Dr. Mustafa GÖK
Director
Institute of Natural and Applied Sciences

ABSTRACT

MSc THESIS

PYRAMID SELF ORGANIZING MAPS (PSOM): A NEW APPROACH TO ANALYZE LARGE DATA SETS

Drilon JAHIRI

ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES
DEPARTMENT OF PLANT PROTECTION

Supervisor : Assoc. Prof. Dr. Mustafa ORAL
Year: 2015, Pages: 93
Jury : Assoc. Prof. Dr. Mustafa ORAL
: Assoc. Prof. Dr. Selma Ayşe ÖZEL
: Assoc. Prof. Dr. Oya H. YÜREĞIR

The aim of this study is to develop a new method capable of processing and analyzing large data sets, by less time consumption and furthermore, to include extra features with the ability of visualizing the correlation between attributes of the analyzed data sets. Another additional goal that the study aims to fulfill is, designing and developing a software package that provides new features of data analysis and data visualization techniques, with the main purpose of interactively analyzing and visualizing the correlation between data attributes.

Data analysis is the process of reading, cleaning, correcting, transforming and modeling data with the purpose of subtracting important information. Analyzing big data is an actual challenge of data analysis techniques. Self-Organizing Maps is an ANN algorithm that successfully processes data sets and visualizes the correlation between attributes. However, it is very slow when processing huge amount of data.

In this study, a Pyramid Self-Organizing Maps (PSOM), based on Self-Organizing Maps algorithm, is proposed. The suggested method successfully overcomes SOM, in terms of time consumption and quality of results. PSOM is a hierarchy-based approach, where within a training process it constructs and trains several maps of several levels (several map-sizes). To further improve the performance of proposed algorithm, Batch version of PSOM is parallelized and adapted to work in multi-core environments. Parallelized Batch - PSOM (PB-PSOM) version significantly overcomes both, PSOM and SOM, in terms of execution time and in terms of quality. A software package, which implements SOM, PSOM and PB-PSOM, and includes extra user-interactive features to analyze data sets, is another contribution of this study.

Suggested algorithms are tested and compared with results of classical SOM using different sizes of input data (100, 500, 1000, 5000, 10000, 50000 and 100000 input patterns) and evaluation of results is done in terms of topology preservation, data set mapping and execution time. Results have shown that PSOM decreases the amount of time required to process large data sets and in the same time, it improves the produced results. In the other hand, PB-PSOM significantly outperforms both, PSOM and SOM.

Key Words: Data Analysis, Self-Organizing Maps, Large Data Sets, Pyramid Self Organizing Maps, Parallelized Batch Pyramid Self Organizing Maps

ÖZ

YÜKSEK LİSANS TEZİ

PİRAMİT ÖZ-ÖRGUTLEMELİ HARİTALARI (PSOM): GENİŞ VERİ SETLERİ ANALİZ ETMEDE YENİ BİR YAKLAŞIM

Drilon JAHIRI

ÇUKUROVA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
MAKİNA MÜHENDİSLİĞİ ANABİLİM DALI

Danışman : Assoc. Prof. Dr. Mustafa ORAL
Yıl: 2015, Sayfa: 93
Jüri : Assoc. Prof. Dr. Mustafa ORAL
: Assoc. Prof. Dr. Selma Ayşe ÖZEL
: Assoc. Prof. Dr. Oya H. YÜREĞİR

Bu çalışmanın amacı, geniş veri setlerini daha az zamanda işleyebilen ve analiz edebilen yeni bir metot geliştirmek ve dahası analizi yapılmış veri setlerinin nitelikleri arasındaki korelasyonu görselleştirebilen ekstra vasıfları içinde bulundurmaktır. Çalışmanın yerine getirmeyi amaçladığı bir diğer hedefi ise veri analizi ve veri görselleştirme teknikleri konusunda yeni vasıflar sunan bir yazılım paketi tasarımlamak ve geliştirmektir ki bu da veri özellikleri arasındaki korelasyonunu etkileşimli bir şekilde analiz etme ve görselleştirme asıl amacını gütmektedir.

Veri analizi, instenilen bilgiyi elde etmek amacıyla veriyi okuma, temizleme, düzeltme, dönüştürme ve biçimlendirme sürecidir. Büyük veriyi analiz etme, veri analiz tekniklerinin asıl zorlayıcı kısmıdır. Öz-Örgütlemeli Haritaları (SOM), veri setlerini başarılı bir şekilde işleyen ve veri özellikleri arasındaki korelasyonu görselleştiren bir ANN algoritmasıdır. Ancak, çok miktarda veriyi işlerken oldukça yavaştır.

Bu çalışmada, Öz-Örgütlemeli Haritaları algoritması temel alınmış bir Piramit Öz-Örgütlemeli Haritası (PSOM) ileri sürülmektedir. Öne sürülen metot, zaman sarfiyatı ve sonuçların niteliği açısından SOM algoritmasına üstünlük gösterir. PSOM, farklı seviyelerde (farklı harita-ölçüleri) haritaların oluşturulduğu ve işletildiği bir hiyerarçi-temelli bir yaklaşımdır. Öne sürülen algoritmanın performansını daha fazla arttırmak için, çok-çekirdekli ortamlarda işlem görebilmesi amacıyla PSOM`in yığıtlı (Batch) versiyonu paralelleştirilir ve uyarlanır. Paralelleştirilmiş Yığıtlı-PSOM (PB-PSOM) versiyonu, uygulama süresi ve nitelik açısından gözle görülür şekilde hem PSOM hem de SOM algoritmalarına üstünlük gösterir. SOM ve PSOM algoritmalarını uygulayan ve veri setlerini analiz etmek amacıyla ekstra kullanıcı-interaktif özelliklerini içinde barındıran bir yazılım paketi, bu çalışmanın bir diğer katkısıdır.

Öne sürülen algoritmalar, farklı ölçülerde girdi verisi (küçük, orta ve büyük) kullanan klasik SOM algoritması sonuçları ile test edildi ve kıyaslandi ve sonuçların değerlendirmesi, topoloji koruma (topology preserving), veri seti haritalandırması (data set mapping) ve uygulama süresi dikkate alınarak yapılır. Sonuçlar göstermektedir ki PSOM, geniş veri setlerinin işlenmesi için gerek duyulan süreyi kısaltır ve aynı zamanda elde edilen sonuçları geliştirir. Diger taraftan, PB-PSOM, hem PSOM hem de SOM'dan daha iyi performans gösterir.

Anahtar Kelimeler: Veri Analizi, Öz-Örgütlemeli Haritaları, Geniş Veri Setleri, Piramit Öz-Örgütlemeli Haritaları, Paralelize Toplu Piramit Öz- Örgütlemeli Haritaları

ACKNOWLEDGEMENTS

Foremost, I would have never been able to finish this thesis without the help of Almighty God, who granted us the health, the power and all the blessings of this life, and then without the guidance of my advisor, support from my family and help from my friends. I would like to express my deepest gratitude to my advisor Assoc. Prof. Dr. Mustafa ORAL for the continuous support for my Master thesis, for his caring, patience, motivation, huge knowledge and enthusiasm. His guidance helped me during my research and writing of this thesis. I even could not have imagined of finding a better advisor and friend for my Master research.

My sincere thanks also goes to the government of Turkey and especially to "Turkiye Burslari" for providing me the graduate education grant. I thank my officemates and labmates in department of Computer Engineering of Cukurova University: Dia Abdulkerim, Shahabadin Daneshvar and Mohamed Cirfe, for their perfect behavior, attitude and excellent atmosphere. A particular thank goes to my friend and my roommate Alireza Deljavan Anvari for his support and his endless patience toward me while staying together on the same room. Furthermore, I am extremely thankful to all my close friends in my hometown, for their sincerity and for being great friends.

Particularly, I am very grateful to one of my closest friends, Mu`ad Alqatanany, for his irreplaceable accompany, excellent jokes that made me feel relaxed and especially for his priceless advices. I would also like express my gratitude to my best and closest friend, Bujar Demolli, for his patience, opinions, collaboration and great advises throughout last six years.

A special thank goes to my girlfriend for giving me endless love and moral support that made me feel excellent throughout my last year of research. Last but not the least, from depth of my heart, I would like to thank my parents, my sister and my brother for their love toward me, for being always close to me, helping me and encouraging me.

Finally, I dedicate this thesis to my parents who gave me life and unconditional support throughout my life.

CONTENTS **PAGE**

LIST OF TABLES **PAGE**

LIST OF FIGURES **PAGE**

LIST OF SYMBOLS / ABBREVIATIONS

SOM	: Self-Organizing Maps
PSOM	: Pyramid Self-Organizing Maps
MDS	: Multidimensional Scaling
EOF	: Empirical Orthogonal Function
VQ	: Vector Quantization
ANN	: Artificial Neural Network
BMU	: Best Matching Unit
CPU	: Central Processing Unit
SISD	: Single Instruction Stream, Single Data Stream (SISD)
SIMD	: Single Instruction Stream, Multiple Data Stream (SIMD)
MISD	: Multiple Instruction Stream, Single Data Stream (MISD)
TPL	: Multiple Instruction Stream, Multiple Data Stream (MIMD)
CLR	: Common Language Runtime
PB-PSOM	: Parallelized Batch - Pyramid Self-Organizing Maps
SDOM	: Standard Deviation of the Mean
QE	: Quantization Error
TE	: Topographic Error
RAM	: Random Access Memory
VS.	: Versus

1. INTRODUCTION

Electronic and intelligent devices are designed, and widely used, for helping data analyzers discovering useful information and giving future decisions (or predicting), based on some “experience” earned by analysis of training data sets. This implies the need of analyzing data collected from previous work experience and giving decisions for newly upcoming data sets.

Data analysis is the process of reading, cleaning, correcting, transforming and modeling data with the purpose of extracting important information. By taking advantage of data analysis techniques and available hardware infrastructure, the process of analyzing available data is effortless. Otherwise, one may face difficulties while analyzing data sets on his own. It would take a long time, and in some cases, it would be almost impossible to process and analyze all the amount of data available. It is quite difficult and time consuming even for today`s super computers. That`s why, these days, data analysis techniques are widely employed and continuously improving.

Data analysis techniques can be divided into sub-categories such as Data Mining, Business Intelligence, Predictive Analytics, Text Analytics and Statistical Analysis. Each of these techniques is responsible for analyzing data sets for a certain purpose.

However, to produce complete and more profitable results, after applying data analysis techniques, a data visualization strategy is necessary. Data analyzing techniques without a data visualization technique is somehow, insufficient.

Analyzing big data is another challenge of data analysis techniques. The amount of data is quickly increasing, while the data analysis algorithms are slowly improving and adapting to big data.

Many data analyzing tools are available in market, either as commercial or open source. Some of available tools include visualization techniques, and some other tools implement algorithms for processing big data as well.

After researching the literature, it has been found out that available tools and publications have a common weakness - there isn`t a single tool or package that

processes big data, analyzes them, and in the same time visualizes the correlation between attributes of these data. Commercial tools use the actual infrastructure of data mining or ANN algorithms, and thus, they are slow on processing big data.

Self-Organizing Maps is an algorithm that successfully processes data sets and visualizes the correlation between attributes. However, it is very slow when processing huge amount of data. Even though it visualizes the correlation between attributes, it does not provide any correlation-hunting feature. Therefore, if the number of dimensions of a data set is huge, one will have difficulties trying to analyze the correlation between attributes (dimensions).

Thereupon, the aim of this study is to find a way of processing huge data sets by consuming less amount of time, keeping good quality of data analyzing results, and in the same time, visualizing the correlation between attributes. Another additional goal that the study aims to fulfill is to design and develop a system that provides some extra features to facilitate the analysis of correlation between data attributes.

A modification of Self-Organizing Maps algorithm, in order to process huge data sets much faster, by improving the quality of results, is the first contribution of this study to the existing knowledge. In addition to this, design and development of a software package based on SOM algorithm, which includes extra user-interactive features dedicated to analyze the correlation between data attributes, is another contribution of this research.

This study, has proposed a pyramid SOM approach, where traditional way of training SOM is modified to a hierarchy-based training approach. The network is trained in several levels, and for each level, the number of neurons increases exponentially. In this way, the algorithm constructs very small maps in the first levels, and gradually the map size increases in other levels (think of a pyramid containing maps inside, parallel with its base). Results from the previous levels serve as a basement for building maps on the next levels. The size of the map increases as the number of levels increases. At the very last level, the map is supposed to be perfectly organized. This hierarchy-approach training leads to a well topologically

ordered map at the last level, and the most important thing, a training process that consumes less amount of time.

Besides, to further speed-up this hierarchy-based approach, a parallelized-batch version is implemented in a processor with four cores. This pyramid approach is tested together with its parallelized version, using different data sets (of different sizes), various map sizes, different subsamples and different error measurements. Clear winner is the proposed method (Pyramid-SOM (PSOM)), which achieves training of the network by keeping the same (or even better) quality of results as in SOM. Furthermore, it attains to accomplish its work process in a very short time.

2. PREVIOUS STUDIES

There are several classical methods capable of visualizing the distribution of large data sets, by mapping or projecting high dimensional datasets to a two-dimensional space. Multidimensional Scaling (MDS) (Torgerson, 1952; Leeuw & Heise, 1982) is such a method, and its widely implemented version is known as Sammon`s projection (Sammon, 1969). For large data sets, these classical algorithms and their mappings are time consuming because of heavy computations.

In order to process, analyze, and visualize the correlation between the attributes of large data sets, it is suggested that Kohonen Self-Organizing Maps type of unsupervised neural network offers the best level of performance (Openshaw & Turton, 1996).

Usually, attempts to analyze large data sets, in a lower amount of time, using Self-Organizing Maps focuses on design and development of parallelized code for parallel environments. In 1996 (Openshaw & Turton), a parallelized version of SOM was designed to classify large spatial data sets. They demonstrated its performance on the Cray T3D parallel super computer (installed in Edinburgh in 1994), which on that time was considered as the seventh fastest supercomputer. The main problem of this algorithm was its focus only on spatial datasets.

In 1999, a scalable parallel implementation of Self-Organizing Maps suitable for Data Mining applications is described (Lawrence, Almasi, & Rushmeier, 1999). It involves clustering or segmentation of large data sets. The data sets for this proposed method are derived from analyses of customer spending patterns. This parallel algorithm is based on another version of SOM, namely Batch SOM (see Section 3.1.7). Batch SOM is a suitable version for parallelizing SOM as it updates the weights of the nodes only at the end of each epoch (there is no inter-node communication until the end of an epoch). This algorithm`s performance has been measured in a SP2 parallel computer, for two retail datasets and a publicly available census dataset. It achieves a linear speedup of Batch SOM algorithm.

Another SOM-based approach is developed in 2000 (Kohonen, Kaski, Lagus, & Salojärvi, 2000). This methodology, called WEBSOM, is specialized for exploring

massive document collection. Textual documents are arranged onto a map, which has useful features for massive document collection searching. Focus of this study is based on reducing the order of magnitude of an application for processing massive document collections, finding a new method for forming statistical models of documents, finding some “shortcuts” for SOM algorithm such as improvements on a rapid construction of large document maps, implementing another map-initialization method and another winner-searching technique.

In 2006, another method for a parallel implementation of SOM is proposed (Silva & Marques, 2007). It is a hybrid method, in the sense that it implements two widely used methods in parallel approaches of SOM, network-partition and data-partition. This method is especially effective when trying to construct large maps. It achieves a topological ordering of the map in first few iterations by implementing data-partitioning method, using batch version of SOM (Batch SOM is suitable for data partition approaches). This method achieves an average speed up of 1.27 over classical batch data-partition methods, and in the same time, it keeps the topological information of the maps. The main disadvantage of this method is that it cannot achieve same results as in original batch algorithm.

Other studies mainly focus on growing the number of neurons in the map. Firstly, they start with a small number of neurons, and while training-phase is going on, they increase the number of neurons in the map. This leads to a big map construction at the end of training phase and a significant reduction of the amount of time necessary to train a big map. Such map-growing algorithms are shown in 2007 (Tachibana & Furuhashi, 2007) and in 2009 (Tachibana, Sugimoto, & Shiogama, 2009). In both of them, a growing self-organizing map is proposed. Both are applicable to spherical visible spaces. The second one is applied on huge climate data as an additional alternative to empirical orthogonal function (EOF) analysis, which is similar to principal component analysis.

3. MATERIALS AND METHODS

3.1. Materials

3.1.1. Vector Quantization

Vector quantization is a lossy data compression technique used for large data sets. In early days, VQ has been considered as a challenging issue due to the need of multi-dimensional integration (Gersho & Gray, 1991). Later on, in 1980, (Linde, Buzo, & Gray, 1980) (LBG) proposed a VQ algorithm based on a training sequence. This technique overcomes the problem of necessity for multi-dimensional integration.

Given a data set, where each pattern is a codevector, VQ finds a codebook (red dots in Figure 3.1) and a partition (blue lines in Figure 3.1) with the smallest average distortion. For a given codevector x from the data set, the process of building the codebook is achieved by simply computing the distance from codevector x to each centroid c_i in the codebook. The closest centroid c_b (red dots in Figure 3.1) to the codevector x is then moved toward codevector x.

Any available distance metric can be used to measure the distance between the codevector x and the centroids in the codebook. A simple algorithm for Vector Quantization is:

1. Randomly take an input pattern from the data set
2. Move the closest centroid from the codebook toward the input vector, by a small distance
3. Go to Step 1 and repeat until stopping criteria

As a stopping criteria, number of iterations, no changes (no more movements of the centroids) or changes in total distortion is below a threshold can be used.

Vector quantization is based on the competitive learning approach, which defines it as closely related to the self-organizing maps model. Vector quantization is

useful for lossy data compression, lossy data correction, video and audio codec building and pattern recognition. It is used in as lossy data correction technique in problems of recovering missing data from some attributes of data sets. In such usage, VQ predicts the value of the missing data based on the values of other data in the same attribute, by assuming that they will have a similar value as the group`s centroid.

VQ is also used in data compression field. As shown in Figure 3.1, 4096 points (green dots) in a two-dimensional space, after applying vector quantization technique are represented by only 16 points (red dots). Saving 4096 two-dimensional points in a memory of a machine requires exactly 2^{12} bits, or 12 bits per each point. VQ technique compresses 4096 points by representing them with just 16 points (representatives). Each representative represents a sub-set of the original data set (4096 points). Furthermore, these new representatives require only 2^4 bits to be placed in a memory.

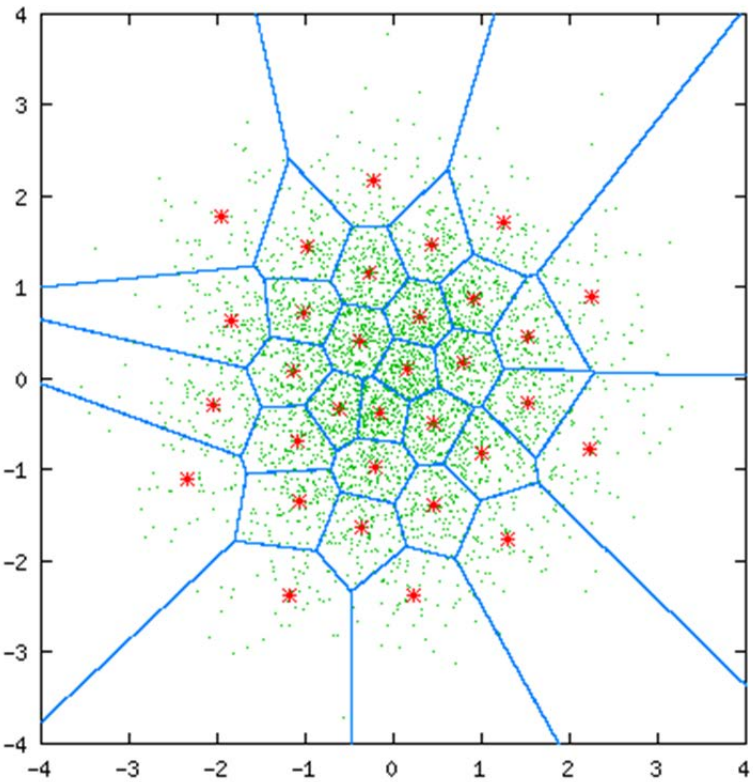


Figure 3.1. 4096 green points represented by 16 red points in a two-dimensional space.

3.1.2. Self-Organizing Maps

Self-Organizing Maps (SOM) is an artificial neural network (ANN) algorithm that fulfills its own tasks in an unsupervised learning approach to produce a low-dimensional (typically two-dimensional) representation of the input space of the data set. This discretized representation is called a map. SOM can be used in the same time for clustering the multidimensional data set and then projecting and visualizing it onto a lower-dimensional space, similar to multidimensional scaling (Duch & Naud, 1996). In addition, unlike other ANN algorithms, to preserve the topological characteristics of the input space, SOM uses a neighborhood function. This makes SOM different from other ANN algorithms. Teuvo Kohonen, a professor from Finland, first defined the algorithm as an ANN and it is often called a Kohonen maps (Kohonen, 2001, pp. 105-116).

A Kohonen map contains components called nodes or neurons. Each node is represented by a weight vector of the same dimension as the dimension of the data set and a position in the map space. Nodes in the map space are usually arranged in a rectangular or hexagonal grid where each rectangle or hexagon in the grid represents a node component. SOM algorithm will arrange these nodes in such a way that the final topological order of the nodes in the map will be a representation of the input space or a mapping from a higher dimensional input space to a lower dimensional map space, namely, maps. The number of maps in the output space is equal to the number of attributes in the input space. Each attribute from the input space is mapped onto a map in the output space. The main part of the process of mapping a vector (an input data) from data space onto the map is accomplished by finding the node with the closest (smallest distance metric) weight vector to the data space input vector.
The SOM algorithm can be summarized on the following four phases:

- Initialization: The weights vectors of the nodes can be initialized in several ways, but two ways, random initialization and linear initialization (Kohonen, 2001, pp. 142-143), are widely implemented. Random initialization of the weights of the nodes can be performed by either generating random values

for each node, or by selecting random vectors from the input space. On the other hand, linear initialization, first determines the two largest (with the largest eigenvalues) principal component eigenvectors, and then these eigenvectors are used to span a linear two-dimensional space. The linear initialization implies a faster convergence of the algorithm as it starts from, already, a well ordered map, and then it just, conditionally saying, fine tunes the map.

- Competition: For each input vector, all the nodes calculate the distance metric from the input pattern to their weight vectors, respectively. The smallest value (distance) is then awarded as the winner, or namely, Best Matching Unit (BMU).
- Collaboration: The winning neuron defines its own neighbors, that are available and excited (close enough) to collaborate with it, and hereby performs the cooperation with them. The neighbors of the BMU are determined by a neighborhood function which, first, is as big as the map itself, with the BMU as the center, but by the time, it shrinks iteratively, until it reaches the size of just one node.
- Adaptation: Neurons within the neighborhood of the BMU adjust themselves by modifying their own weight vector, according to their distance to the BMU and the value of the input pattern mapped to the actual BMU. Neighboring neurons will be affected from the values of the input pattern and from a coefficient proportional to the distance to the BMU. Their adjusted weights will tend to resemble to the input pattern.

3.1.2.1. The Competition Process

In a D dimensional space (i.e. data space containing D attributes), the input patterns from the input space can be written as $x = \{x_i : i = 1, 2, ..., D\}$, while the weight vectors that connect the input units i and the node j in output layer (space)

can be written as $w_j = \{w_{ji} : j = 1,...,N; i = 1,...,D\}$, where N is the total number of nodes in the output space.

In the competition phase, only one neuron will be the winner of the competition. The node, whose weight vector comes closest to the input vector, is declared as BMU.

Several distance measuring metrics are implemented when computing the distance between input patterns and weights of nodes, but those finding a wide implementation are, Euclidian Distance and Manhattan Distance.

Using Euclidean Distance metric, the discriminant function that defines the distance between the input vector x and the weight vector w_j for each neuron j, has the following form:

$$d(x) = \sum_{i=1}^{n} (x_i - w_{ji})^2 \tag{3.1}$$

Furthermore, the winning neuron for an input pattern is determined by the following condition:

$$i(x) = \arg\min_j (d(x)) \tag{3.2}$$

Where, $i(x)$ is the index of the neuron closest to the input vector x. Thus, the continuous input space can be mapped to a discrete output space of nodes by the process of competition between the neurons.

3.1.2.2. The Collaboration Process

There is a neurobiological evidence for lateral interaction within a set of excited neurons (Haykin, 1999, p. 449). When a neuron fires it excites its neighbor neurons. Neighbors in the immediate neighborhood are excited more than the neurons located further away. As an implication of this statement, there is a

topological neighborhood around the winning node i, which fences in the set of excited neurons (Haykin, 1999, p. 449).

Let h_{ji} define the topological neighborhood centered on the BMU i and let this neighborhood fence in the set of excited neurons. If the exited neurons (collaborating) are denoted by j and if d_{ij} denotes the distance between winning neuron i and excited neuron j, then the function that best fits the requirements of topological neighborhood of SOM is the Gaussian function:

$$h_{j,i} = \exp(-\frac{d_{j,i}^{2}}{2\sigma^{2}}) \tag{3.3}$$

The parameter σ, as shown in the figure 1, is the width of the topological neighborhood function $h_{j,i}$ in the output space.

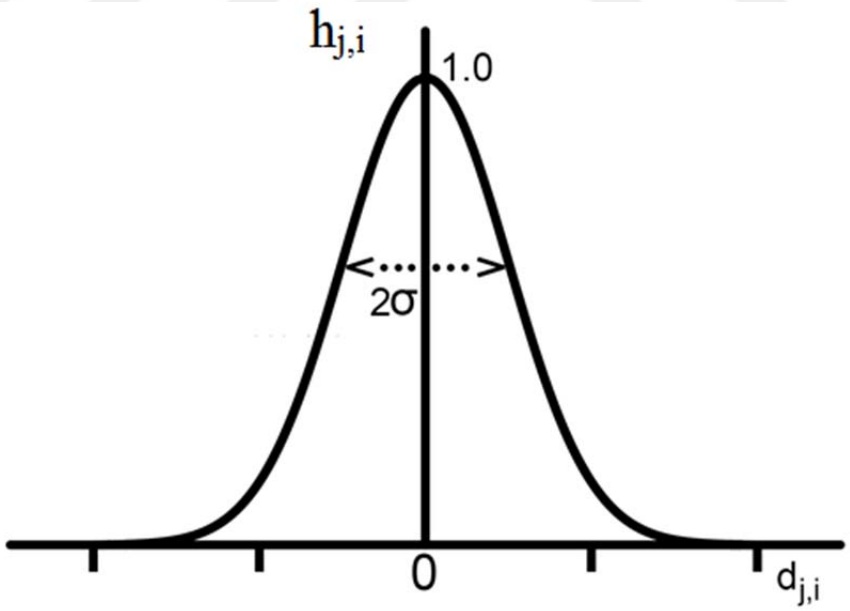


Figure 3.2. Gaussian neighborhood function prior to SOM`s usage

Main and the most important characteristics of this topological neighborhood are:

- It reaches its maximal value at the winning neuron
- It is symmetrical about the winning neuron
- As the distance goes to infinity, it decreases monotonically, shrinking to zero.
- It is translation invariant (i.e. independent of the location of the BMU)

Such a Gaussian topological neighborhood is more qualitative than the rectangular topological neighborhood. SOM algorithm converges faster with the use of Gaussian topological neighborhood (Lo, Yu, & Bavarian, 1993; Erwin, Obermayer, & Schulten, 1992).

A useful feature of SOM is that the topological neighborhood shrinks by the time. To achieve this characteristic of SOM, the spread factor σ should decrease with time. A widely used time dependent function is an exponential decay:

$$\sigma(t) = \sigma_0 \exp(-\frac{t}{\tau_0}) \qquad t = 0,1,2,..., \tag{3.4}$$

where σ_0 is the initial neighborhood value of σ, and τ_0 is a time constant. When considering time, equation (3.3) will take the form as follows:

$$h_{j,i}(t) = \exp(-\frac{{d_{j,i}}^2}{2\sigma^2(t)}) \qquad t = 1,2,3,..., \tag{3.5}$$

where $\sigma(t)$ is defined by equation (3.4), and t is the number of iterations.

3.1.2.3. The Adaptation Process

For the network of SOM algorithm, to be a self-organized one, SOM involves the process of learning (self-learning), which actually is a kind of adaptive process. By this adaptation, output space becomes self-organized, and thus such an

organization is actually a representation (mapping) of input space in the output space. Technically, this is achieved by changing the weight vector w_j in order to resemble to input vector x. However, not only the winning neuron gets its own weights updated, but its neighbors will be updated as well, even though not as much as the winning neuron. In a discrete-time form, the equation used to update the weight vector is:

$$w_j(t+1) = w_j(t) + \eta(t)h_{j,i}(t)(x - w_j(t)) \tag{3.6}$$

where $w_j(t)$ is the weight vector j at time t, $w_j(t+1)$ is the updated weight vector at time $t+1$. Equation (3.6) is applied to all the neurons in output space and it modifies the weights of neurons that are neighbor of winning neuron *i*. In other words, the equation has the effect of moving the winning neuron with the weight w_i (together with its neighbors) of BMU i toward the input vector q.

In the equation (3.6), another heuristic parameter is applied, the learning-rate parameter η. Learning –rate parameter is a time-varying parameter which starts with an initial value of η_0 and then it decreases with increasing time t. Equation (3.7) makes this possible:

$$\eta(t) = \eta_0 \exp(-\frac{t}{\tau_1}) \qquad n = 0,1,2,..., \tag{3.7}$$

where τ_1 is again another time constant of the algorithm.

During the process of adaptation, two phases are followed (Haykin, 1999, pp. 452-453):

- Ordering phase: During this phase a rough topological ordering takes place, while the learning parameter and the number of iterations have a significant

importance. As suggested in (Haykin, 1999, pp. 452-453) learning rate $\eta(t)$ should begin with a value around 0.1. The neighborhood function, in the beginning, should fence in all the neurons in the map and then with time it should reduce this set of neighbors

- Convergence phase (fine-tuning): The learning parameter $\eta(t)$ should be kept around 0.01 and it should never be zero; otherwise, it may stack to local minima. The neighborhood function $h_{j,i}$ should be small enough to include only its immediate neighbors; later it can decrease to zero (no neighbors, just itself) (Haykin, 1999, pp. 452-453).

3.1.2.4. SOM Summary

The aim of SOM is to map a high-dimensional input space to a lower discrete dimensional space. This is achieved by arranging a set of neurons in a grid in output space.

SOM algorithm can be summarized in the following phases (Haykin, 1999, p. 454):

1. Initialization: Random values are chosen to initialize the weights w_j of neuron j.
2. Sampling: Draw a sample vector x from the input space and fed it into the network.
3. Matching: Find the winning neuron i with weight vector w_i closest to the input vector x.
4. Updating: Apply weight update equation
   $w_j(t+1) = w_j(t) + \eta(t) h_{j,i(x)}(t)(x - w_j(t))$.
5. Looping: Continue with step 2 until the map does not change anymore.

Figure 3.3 illustrates SOM network and the relationships of input/output spaces.

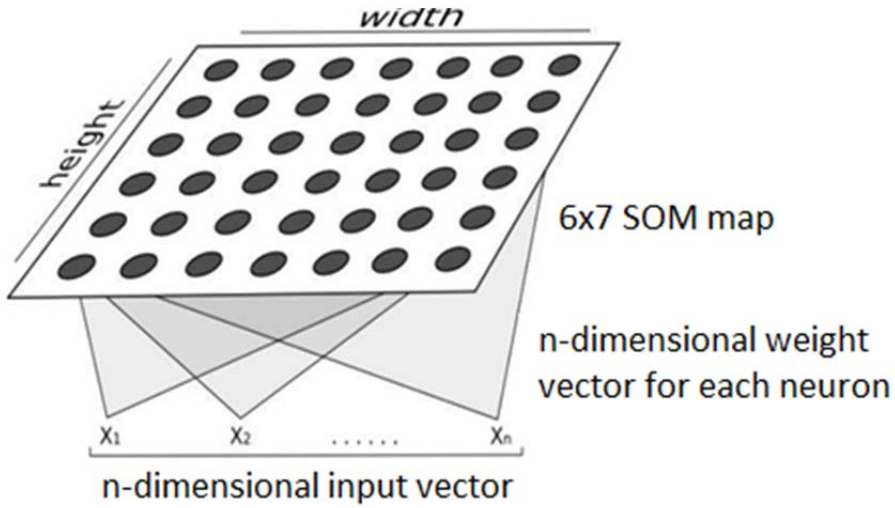


Figure 3.3. Relationship between neurons in the output space and the input x from continuous input space.

3.1.3. Euclidean Distance

In mathematics, the Euclidean Distance is the "common" distance between two points that could simply be measured with a ruler. Euclidean Distance or Euclidean Metric is given by the Pythagorean formula. By using this formula as distance, Euclidean space (or even any inner product space) is actually a metric space. The associated norm is called the Euclidean norm.

In Cartesian coordinates, if $p=(p_1, p_2, ...p_n)$ and $q=(q_1, q_2, ...q_n)$ are two points in Euclidean n-space, then the distance from p to q, or from q to p is given by:

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + ... + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \qquad (3.8)$$

3.1.4. Manhattan Distance

The Manhattan distance function computes the distance that would be traveled to get from one data point to the other in a grid space. The Manhattan distance between two points is the sum of the differences of their corresponding coordinates. In a grid space, their corresponding coordinates are actually their node position in the grid.

The formula for this distance between a point $p=(p_1, p_2,...p_n)$ and a point $q=(q_1,q_2,...q_n)$ is:

$$d(p,q)=d(q,p)=\sqrt{|q_1-p_1|+|q_2-p_2|+...+|q_n-p_n|}=\sqrt{\sum_{i=1}^{n}|q_i-p_i|} \quad (3.9)$$

Where n is the number of variables, and p_i and q_i are the values of the ith variable, at points p and q respectively.

The following figure illustrates the difference between Manhattan distance and Euclidean distance:

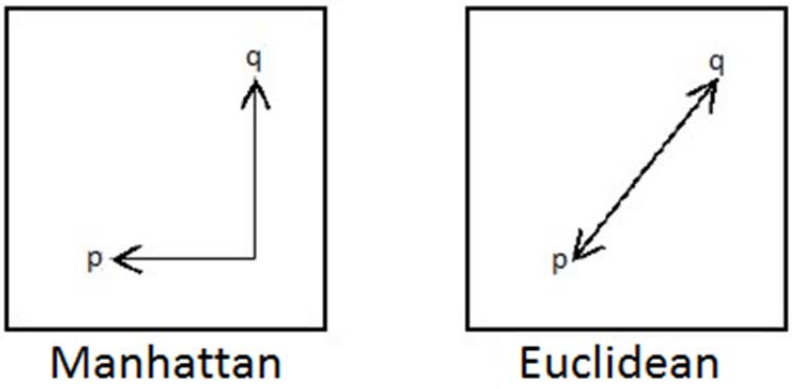


Figure 3.4. Difference between Manhattan Distance and Euclidean Distance

3.1.5. Neighborhood Functions

Mathematically, neighborhood is a significant and basic concept in topological spaces. Speaking in terms of ANN algorithms, and especially in terms of SOM, neighborhood around a point (winning neuron) is a set, which fences in some other neurons close to winning neuron. Furthermore, the point in the center (winning neuron) can affect other neurons inside this set and push them to change their state (weights) closer to the state of winning neuron.

Speaking in terms of theoretical definitions, a neighborhood of a point p in a topological space X is a subset S of X, where X has an open set V that contains p, where $p \in V \subseteq X$ (Kelley & John, 1975, pp. 26-27).

In the case of Self-Organizing Maps, different neighborhood functions can be used, but widely, neighborhood range is determined using a Gaussian kernel around the winner neuron, or a rectangular neighborhood function (Figure 3.5). In terms of processor calculations of a machine, Gaussian function, as in the equation (3.5) , is computationally more expensive as the exponential function has to be computed by the processor. This function can be well-approximated using rectangular function, which requires less calculation, but in the same time, it increases the number of neurons to be updated in the update phase of Self-Organizing Maps.

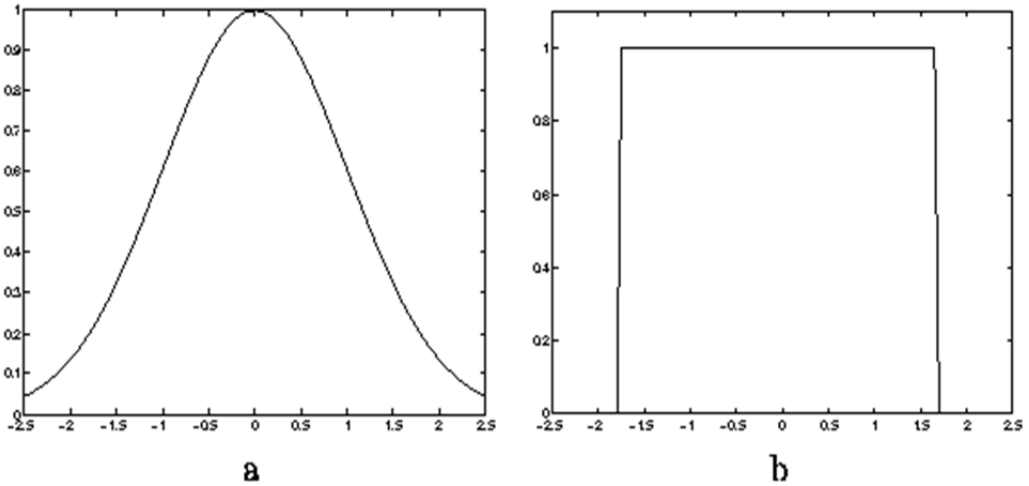


Figure 3.5. Gaussian function and Rectangular function

3.1.6. Parallelization and Multi-core Programming

3.1.6.1. Data Parallelism

When processing very large data sets, a typical analysis may consume a lot of CPU time, but when splitting this data set across, maybe thousands of machines, the analysis may need just several hours to accomplish (Pike, Dorward, Griesemer, & Quinlan, 2003). In other words, data parallelism (Mann & Haykin, 1990; Ceccarelli, Petrosino, & Vaccaro, 1993; Myklebust & Solheim, 1995; Ienne, Thiran, & Vassilas, 1997) is a parallel approach of processing and distributing data sets across several processors (cores), in a parallel environment. After the split data set is computed separately, it is again arranged into a single data set. Data Parallelism is machine and algorithm independent. It can be applied in all algorithms, but if the algorithm in its original form is not appropriate for data parallelism technique, then it should be adapted to fit data parallelism approach.

This technique of having one processor per data element, changes the way one thinks as programming approach and design has to change in order to fit a parallel environment (Pike, Dorward, Griesemer, & Quinlan, 2003). In a parallel environment the algorithm should efficiently take advantage of all processors; classical programming approaches has to be adapted to parallel programming approaches, and this is a challenge for a classical programmer.

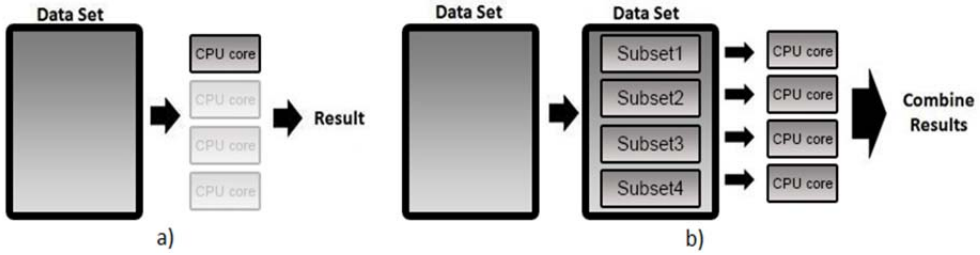


Figure 3.6. a) In traditional programming approaches, a data set is processed by a single CPU core. b) Using data parallelism technique original data set is splitted into subsets, where each subset is given to a CPU core.

3.1.6.2. Multi-core Programming

In today's technology, increasing the number of transistors does not increase the computing capability of a system. One of the new techniques used to improve the performance of a system is the development of multi-core processors. In multi-core processors, two or more processors are used to increase efficiency and enhance performance (Saleem, IkramUllah Lali, & Nawaz, 2014). Multi-core architecture can be used in both, personal systems and embedded systems. According to Flynn`s taxonomy, computer architecture is classified into four classes as follows:

- Single Instruction Stream, Single Data Stream (SISD)
- Single Instruction Stream, Multiple Data Stream (SIMD)
- Multiple Instruction Stream, Single Data Stream (MISD)
- Multiple Instruction Stream, Multiple Data Stream (MIMD)

Considering these four classes, only the fourth one (MIMD) is able to execute multiple instructions simultaneously in combination of working in separate and independent data stream (He, Fang, Govindaraju, & Luo, 2008). This ability makes MIMD as the most appropriate for parallel computing.

Traditional methods of programming do not take advantage of the capacity of CPUs. They use one core of a CPU while the other cores are in a idle state. Parallel programming approach overcomes this weakness by adapting in a multi-core environment, thus by using the whole capacity of a multi-core processor. When developing systems to fit a parallel programming environment, the developer should always follow eight rules for parallel programming (Reinders, 2007):

- Think parallel: Decide for best parallel approach before other designs or implementations, and organize your thinking to express the best parallel approach.
- Program using abstraction: Programmer`s approach should always be focused on writing parallel code instead of writing code that tries to manage threads

or cores. Working directly with cores gives a hard time to the developer to write, debug or handle code. Instead, programmer should take advantage of libraries available in almost all programming languages, such as Task Parallel Library in .NET 4.0.

- Program in tasks, not threads: The mapping of tasks to threads or processor cores is the responsibility of to the library of the programming language you are using. In .NET 4.0, a developer just has to write the task-based code, and the process of mapping code to threads or cores of the processor is carried out by TPL library.
- Make concurrency configurable: A system that works in parallel should also be able to work in just one thread. In this way, when the physical constraints of a machine do not provide a parallel environment, the system should still be working in a single core. In addition, this programming technique allows the programmer to; easily debug the system in a single thread machine.
- Stay away from using locks: A parallel program which uses locks is slow, has a low scalability and is a source of bugs. A parallel approach of a system should be designed in a way that will avoid the usage of locks, so locks will be used only out of the system.
- Use tools and libraries designed to handle concurrency: Most tools are not designed for parallel programming, so the decision of selecting a tool should be careful and preferable a thread-safe library that is designed to handle parallelism itself, should be selected.
- Use scalable memory allocators: TPL in .NET 4.0 provides scalable memory allocators in Common Language Runtime (CLR). TPL uses these allocator automatically (under the hood) when a parallel system works with tasks and threads. Using scalable memory allocators speeds up a system and better utilizes caches within thread.
- Plan early for scalability in the future: As time passes, new processors are produced (or bought), with increased number of cores. A parallel program should be scalable in the future. It should have the ability to increase its workload as the number of cores increases in the future. For this reason, a

parallel system should be designed a priori to take advantage of hardware improvements (and updates).

However, developing a code that effectively takes advantage of a parallel environment is not easy (Openshaw & Turton, 1996).

3.1.7. Batch Self-Organizing Maps

Another version of SOM algorithm (Kohonen, 2001) is an offline variant of the algorithm, namely Batch Self-Organizing Maps (Offline algorithm) (Kohonen T. , 1993).

The weight update equation (3.6) referred as "online" as the weight updates occurs after each input pattern presentation. Unlike online SOM, in Batch SOM algorithm weight vectors are updated just after all input patterns are presented to the network. In other words, weight update phase starts only at the end of each epoch. This implies a modification of equation (3.6) as follows:

$$w_j(t_f) = \frac{\sum_{t'=t_0}^{t'=t_f} h_{j,i}(t')x(t')}{\sum_{t'=t_0}^{t'=t_f} h_{j,i}(t')} \tag{3.10}$$

where t_0 and t_f stands for the beginning and end of the present epoch, respectively. $w_j(t_f)$ are the weight vectors computed at the end of the present epoch. This means that the summations of weights for each input pattern are accumulated during one epoch. Equation (3.10) actually defines k-means algorithm, when there is no neighbor (Fort, Letremy, & Cottre, 2002). The winning node is determined using

$$d_j(t) = \| x(t) - w_j(t_0) \|^2 \tag{3.11}$$

$$d_c(t) = \min_j d_j(t) \tag{3.12}$$

where $w_j(t_0)$ are the weight values calculated at the end of previous epoch. Neighborhood function $h_{j,i}$ is computed using equation (3.5), but using the winning node derived from equation (3.12). The other features and equations of Batch Self-Organizing Maps are the same as in the online version of the algorithm. Batch version of the algorithms has several advantages over the online version. There is no dependency on the order that input patters are presented to the algorithm, as the weights are only updated at the end of one epoch. There are two main differences between these two versions of the algorithm – the late updating and the averaging of data vectors (Ncker, Mrchen, & Ultsch, 2006). All other comparisons between online SOM and offline (batch) SOM are discussed in (Fort, Letremy, & Cottre, 2002) and (Ncker, Mrchen, & Ultsch, 2006).

3.1.8. Triangular Initialization Approaches (Linear Interpolation Between Two Points)

Linear interpolation is used to determine a value of a point in the straight line between two points, (x_0, y_0) and (x_1, y_1) (see Figure 3.7) (Meijering, 2002). In terms of grid space, linear interpolation is used to determine the value of a grid node (weight) by using known neighbor values.

To determine the value y in the straight line, for a given value x in the interval (x_0, x_1), equation (3.13) is used.

$$y = y_0 + (y_1 - y_0)\frac{(x - x_0)}{(x_1 - x_0)} \tag{3.13}$$

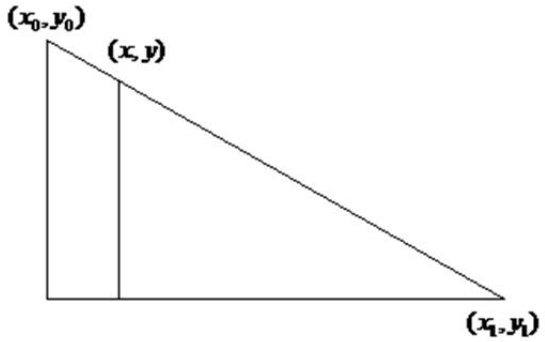


Figure 3.7. Linear interpolation between two points.

Interpolation is useful when initializing nodes of the map in SOM. First two nodes similar to points (x_0, y_0) and (x_1, y_1) are determined and then all other nodes in the straight line between these two points, are initialized using a modified version of equation (3.13). For more details related to implementation of linear interpolation in map-initialization, read section 3.2.2.

3.2 Methods

3.2.1 Pyramid Self -Organizing Maps

Pyramid Self-Organizing Maps (PSOM) aims to reduce the amount of time necessary to train big maps with large data sets. In addition, it maintains, and even improves its results comparing to results obtained from classical SOM algorithm.

Main issues of SOM algorithm arises when calculating all distances from each input pattern to every neuron in the map and heavy computations when updating weights of neurons within the neighborhood of winning neuron. In huge maps, and for huge data sets, these two problems highly affect the performance of the algorithm.

Pyramid SOM focuses on reducing the number of calculation necessary to find the distances from each input pattern to every node and reducing the number of neurons to be updated. To achieve this, PSOM implements a hierarchy-based training approach. It consists of several levels where each level of PSOM is actually a self-

organized map of a particular size. The very first level contains a map of a very small size - several neurons. Second level of PSOM builds a bigger map based on the map trained in the first level. Third level of PSOM constructs another map, which is even bigger than the maps of first and second level. The map in the third level is constructed using the information from the maps in two previous levels. In other words, the map of a certain level always uses the information given from maps in previous levels. Thereby, at the last level, the map is very big and it is topologically well ordered; because it always tries to correct the quality of maps from previous levels. Thereupon, this method is called Pyramid SOM. The pyramid starts with a small level (map) at the top, while the other levels increase their size as it goes downwards to the end of pyramid (see Figure 3.8).

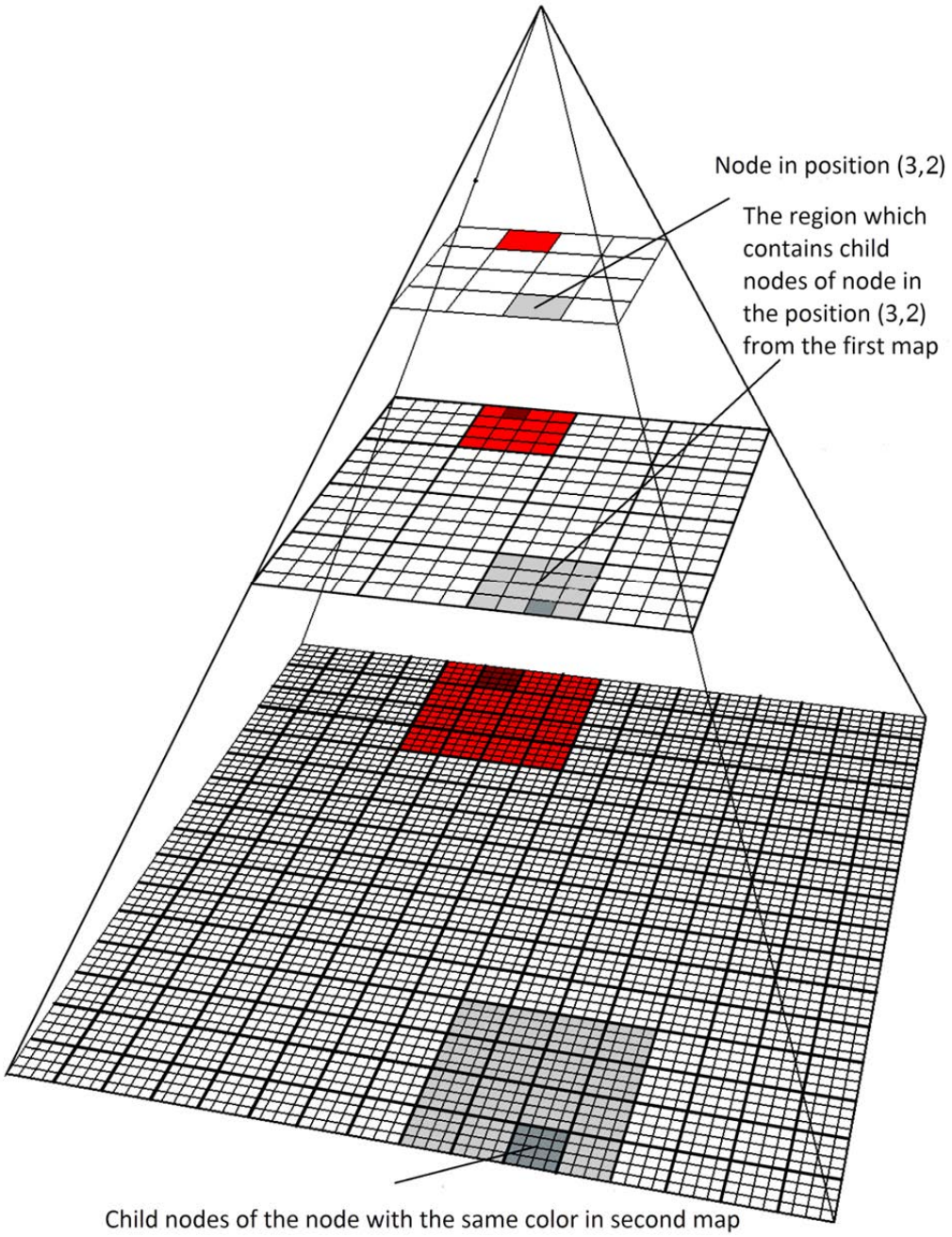


Figure 3.8. 3-level PSOM. Connection between maps is followed from first map to last map. Each neuron is a parent of a group of neurons in the next map

To understand more easily how PSOM works, a PSOM with $16 = 4x4 = 4^2$ neurons in the first map is explained. Otherwise, the logic is the same for different number of neurons in the first map.

In the first level, a subsample of the data set is used to train an SOM network, which contains a map of just 16 nodes (a $16 = 4x4 = 4^2$ map size). Nodes are initialized using random initialization technique (random numbers in the interval $[0,1]$). After accomplishment of training process, each input pattern from the original data set is associated to a node in the map. Precisely, each input pattern is represented by the closest neuron (BMU or winner neuron) of the map. Neurons on this level are not only representatives of a group of input patterns, but at the same time, each of the neurons will be a parent of a group of nodes in the next level of pyramid. Hereafter, we will refer to terms "level" and "map" as the same entity. Whenever term "map" is used, it means that this map belongs to a certain level. First map is trained as a usual classical SOM.

The next step involves the process of initialization and construction of the next map, downwards in the pyramid (second map). Each neuron in the first map will be a parent of $4x4$ neurons in the second map. For each neuron in the first map, there are $16 = 4x4 = 4^2$ child-neurons in the second map. As a definition: for each neuron in a particular level, the number of its child-neurons in the next level is equal to the whole number of neurons in the first map. This definition, when applied to the second level, leads to a second map (second level) which contains $256=16x16=4^2x4^2=(4^2)^2$ neurons. If this explanation is extended to the third map (next level downwards in the pyramid), it means that the third map will have $4096=16x16x16=4^2x4^2x4^2=(4^2)^3$ neurons. Each neuron in the third map will be a child-neuron of a neuron from the second map, while each neuron in the second map is a child-neuron of a neuron in the first map. With the same logic as in the first map, each neuron in the second map is a parent of $16 = 4x4 = 4^2$ neurons in the third map. Extending this explanation to a general case, the number of neurons on each map is determined by the following equation:

$$M_L = M_1^L = ((M_1^R)^2)^L \qquad L = 2,3,... \tag{3.14}$$

where L is the level number, M_L is the number of nodes which we want to calculate in the current level, M_1 is number of neurons in first map while M_1^R is number of rows/columns in the first map. In Pyramid SOM, the number of rows and columns of a map is equal.

Figure 3.8. shows a typical Pyramid SOM containing three levels, where each level is a Self-Organized map. Parents and child neurons are illustrated using the same color. Neurons in the first map can only be parents, while neurons in the other levels are child neurons of their corresponding neurons from the first level and in the same time, they are parents of a group of neurons (their corresponding neurons) in the next level.

3.2.1.1. Initialization Process of Second and Third Map

Neuron`s weights in other levels (except first level) are initialized using weights of neurons from the previous map. Initialization of a map is accomplished using a modified version of Linear Interpolation method (see Section 3.1.8).

Second map is logically divided into 16 regions, where each region belongs to a parent neuron from the first map. For example:

- ✓ First region of the second map belongs to the neuron of the first map located in position (0,0)
- ✓ Second region of the second map belongs to the neuron of the first map located in position (0,1)
- ✓ Third region of the second map belongs to the neuron located in position (0,2) on the first map…
- ✓ Finally, following the same methodology, last region in the second map belongs to the neuron of the first map located in position (3,3).

- ✓ Furthermore, as explained previously, each region in the second map contains 16 child-neurons (child neurons of parent neurons from the first map).

In the second map, neurons that lie in the diagonal, between the center of the region of neuron in position (0,0) and the center of the region of neuron in position (3,3) from the first map, are firstly initialized (see Figure 3.9). Next, neurons that lie in the diagonal, between the centers of the region of neurons of the first map located in position (0,3) and (3,0) respectively, are initialized. In this way, all the neurons that lie in diagonals of the second map will have weight values between the weight values of neuron {(0,0), (3,0)}, and {(0,3), (3,0)} of the first map, respectively. When all the neurons that lie in two diagonals of the second map are initialized, then the initialization of other remaining neurons takes place. From now on, every neuron of the second map located in position (r,c) is initialized using the weights of neurons in position $(r,0)$ and $(r,15)$, where r and c stand for the index of rows and columns in the grid, respectively. Furthermore, $r=1,2,...,14$; $c=1,2,...,14$; $(r,0)$ is the first neuron of row r, and $(r,15)$ is the last neuron of row r. Applying the same logic as for diagonals, neurons positioned in row r will have weight values between the first and last neuron of row r.

In a classical SOM, small values are grouped in one corner of the map, big values are grouped in the opposite corner, and all the values in between are spread in the middle of the map (around diagonals of the grid). Considering this distribution of values in an SOM map, this methodology of initialization proposed in this study, leads to a well topologically-ordered map. It keeps big and small values in two opposite corners while distributing mid-values around diagonals of the map. Furthermore, it reduces the amount of work required to train the network since the training process starts in a map that is initialized properly and well topologically ordered. Another characteristic of this initialization is that all child-neurons will have weight values very similar to their parent`s weights. Thereby, training process will be a kind of fine-tuning process, and less iteration will be required to train the map.

3.2.1.2. Training Process of Second and Third Map

While in the first level training process is same as in a classical SOM, training process in other levels starts from the first map. Unlike SOM, when an input pattern is introduced to the network, PSOM does not search in the entire map to find the winner neuron. Instead, it finds the BMU in the first map and then it searches for another BMU in the second map. In the second map, search process is performed only in the child-nodes of the BMU found from the first map (parent neuron). This means that the maximal neighborhood range in PSOM is equal to the maximal neighborhood range of the first map. Neighborhood range is very small comparing to the size of the map. This implies a significant reduction of number of calculations performed to find the winning neuron in whole map, and another significant reduction of calculations when updating weights in the neighborhood of BMU. This workaround is based on our assumption that the BMU of the second map has to be in the region containing child-neurons of the BMU that comes from first map. Our assumption is based on the initialization method that we implement to initialize neurons of a particular map.

The process of training the third map is similar to the process of training the second map. The main difference is that when searching for the BMU, search is not performed directly in the third map. First, a BMU is found in the first map, then another BMU is found in the second map (in the region containing child-neurons of first map`s BMU), and finally the real BMU in the third map is found, in the region containing child-neurons of the BMU from previous map (second map). Weight update process is performed same as in the second map with the same neighborhood range, which is equal to the neighborhood range of the first map.

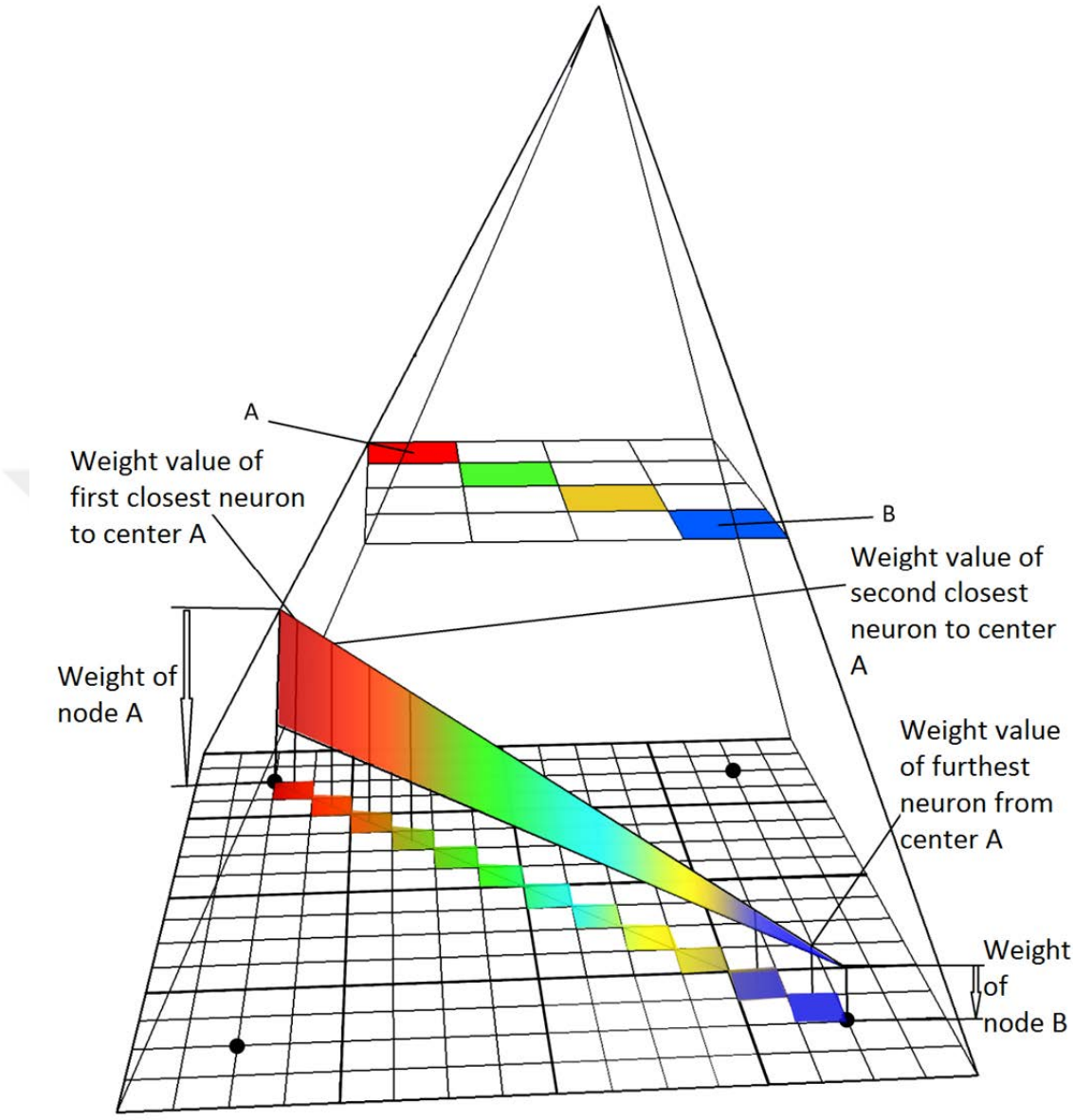


Figure 3.9. Initialization of second map, based on weight values of neurons in the first map.

3.2.2. Initialization Method in PSOM

To initialize neurons of second, third and other maps (if any) in pyramid, PSOM adapts Linear Interpolation method. From Figure 3.9, it is trivial to understand initialization method of PSOM. If the first map is already trained, and all the neurons have their final weight values, neurons in the corners are used to

initialize neurons that lie in the diagonals of the next map (second map). Two neurons in two opposite corners will always have very different weight values; one neuron will have small weight values while the other neuron will have big weight values. This fact provides us the opportunity of employing triangular initialization method. Let neurons in two opposite corners of the first map be A and B, respectively. Let`s also assume that A is the neuron containing big value and B is the neuron containing small value. Furthermore, as explained in this study, neuron A and neuron B are the values of the centers of their child regions in the second map, respectively; each region in the second map is a child region of a node from the first map. The aim is to initialize neurons in the diagonal (of the second map), between child regions of A and B. The triangle in Figure 3.10-b is derived by subtracting weight value of B from the weight value of A.

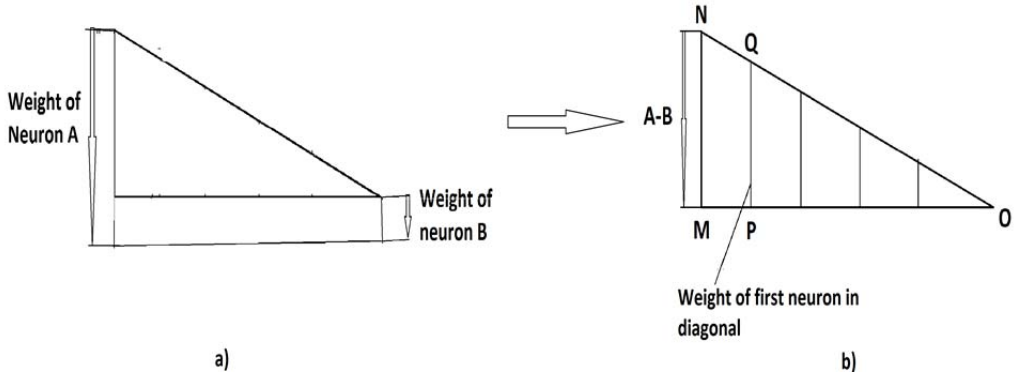


Figure 3.10. Initialization of neurons that lie in diagonal of a map.

By calculating value of $|PQ|$ (3.10 - b), we actually calculate the weight value of the neuron closest to A (value of A is the center value of the first region in the second map); the first neuron in the diagonal between A and B. Mathematically it is known that:

$$\frac{|PO|}{|MO|} = \frac{|PQ|}{|MN|} \tag{3.15}$$

From equation (3.15), $|PQ|$ is determined using the following equation:

$$|PQ| = \frac{|PO| x |MN|}{|MO|} = \frac{(|MO| - |MP|) x |MN|}{|MO|} \tag{3.16}$$

Next, because in the beginning, the weight value of B is subtracted from weight value of A, equation (3.16) slightly changes to:

$$|PQ| = \left(\frac{|PO| x |MN|}{|MO|} \right) + B = \left(\frac{(|MO| - |MP|) x |MN|}{|MO|} \right) + B \tag{3.17}$$

where

- $|MO|$ - distance between A and B
- $|MP|$ - distance between A and first neuron in diagonal (closest to A)
- $|MN|$ - weight value of neuron A

All other neurons in the diagonal are initialized in the same way. Neurons in the next diagonal are also initialized using the same approach. After the initialization of neurons that lie in diagonals, the initialization of remaining neurons is completed row by row, where A and B, from the above example, are actually the first and last neuron of the row, respectively.

This method is employed in third level as well. To initialize the third map, neurons in the corners of the second map are used, by applying the same methodology as described in this section.

3.2.3. Parallelized Batch - Pyramid Self-Organizing Maps (PB-PSOM)

Pyramid SOM processes and visualizes large data sets by consuming less amount of time comparing to SOM. In addition, it succeeds to keep the same quality of maps, and even better than Self Organizing Maps (See Section 4.2). However, this

study went further. Even though PSOM algorithm is faster than SOM algorithm, in this research, another improvement of PSOM itself is proposed.

With the current computer infrastructure where processors contain many cores, it is always better to take advantage of these multi-core computers. Only a few numbers of systems are developed to work in a multi-core environment, even though processors have other available cores as well. Parallelization techniques and multi-core programming (see Section 3.1.6.2) take advantage of parallel environments by using all available computing resources of a processor. Considering this available infrastructure, this study has parallelized PSOM algorithm with the aim of speeding it up.

Usually parallel implementation of ANN algorithms are focused on network partitioning (partitioning network to different processors) or data partitioning (partitioning input data to different processors) techniques (Lawrence, Almasi, & Rushmeier, 1999). Sometimes both, network partitioning and data partitioning are implemented (Silva & Marques, 2007). Both approaches have been tested and comparisons have been shown on more details (Lawrence, Almasi, & Rushmeier, 1999).

In the case of Self Organizing Maps, the main advantage of Network Partitioning methods is the use of the same formula to update its weight vectors (Equation (3.6)). Thereupon, it produces same results as in classical SOM algorithm. However, the disadvantage of Network Partitioning is the necessity of inter-communication of parallel tasks at the end of each iteration, in order to determine the winning neuron. This inter-communication process consumes time since in parallel environments the communication between tasks should be as small as possible. In addition, as explained in section 3.2.1, PSOM algorithm never works with the whole map. Instead, it always works with small number of neurons. This feature makes Network Partitioning useless in case of PSOM.

Furthermore, data parallelization offers a much greater parallel scalability, especially in large data sets, since the data set is split onto different processors (or cores), where each processor performs search and update phase independently. However, last statement stands only for Batch version of SOM since, unlike SOM, in

Batch version neurons are updated at the end of every epoch, using equation (3.10). Just at the end of one epoch, an inter-communication between tasks takes place. This makes data partitioning method suitable for Batch PSOM as well.

Similar to Batch SOM, in Batch PSOM weight vectors are updated at the end of one epoch. The equation used to update weights of neurons is the same as the one used in original Batch SOM (See Equation (3.10)).

Depending on the number of cores of a processor, in Batch PSOM input data set is divided onto subsets. Each subset is given as an input to a task (conditionally saying, to a core). Each task contains a copy of global map and real weight values of global neurons. The process of reading input patterns, feeding to the network, and searching for BMU is the same as in serial PSOM. Instead of updating weights of neurons at each iteration, in parallelized version of Batch PSOM, during one epoch every task accumulates nominators and denominators of equation (3.10) for its own copy of maps. At the end of every epoch, an inter-communication between tasks occurs. The main task, for every neuron in the global map will update global nominators and denominators of the global map according to local nominators and denominators of local tasks. In other words, every task will contribute to the weights of neurons in global map with its locally accumulated nominators and denominators.

3.2.4. Software Package for Data Analyzing

The last but not the least important outcome of this study is to design and develop a software package based on SOM algorithm, which involves extra user-interactive features of data analysis techniques. The main aim of this package is to facilitate the process of analyzing data sets. Its focus is concentrated in visualizing the correlation between attributes of a data set by providing extra user-interactive visualization features.

The software package includes several different modules, listed as following:

- A module for reading input data and selecting data attributes.

- A module based on SOM algorithm to analyze data sets, with selectable and interactive input parameters.
- Visualization techniques for clarification of the mapping from high dimensional input space to discrete output space, in hexagonal or rectangular grids.
- Extra user-interactive features and tools to manipulate trained maps of SOM.

3.1.2.1 Managing Input Data

A module, which provides tools for pre-processing of input data, before feeding it to the network, is firstly developed. The module is able to read the data from xls or xlsx files and load them into the memory. The window for managing data sets is shown in Figure 3.11. Attributes of the data set are firstly shown in their original state on a data grid. User can decide which attributes to use on the software (drag and drop approach). Names and the order of the attribute can also be changed by clicking on the textbox of previously dragged and dropped (selected) attribute.

Attributes can be reselected or modified later on, by clicking in the “Manage Dataset” button. To remove an attribute the user should select it (one by one or all at once) and click on the “Remove” button. To apply changes the user should click on “Apply” button. Furthermore, the normalization technique is applied to input data, in the interval of [0,1].

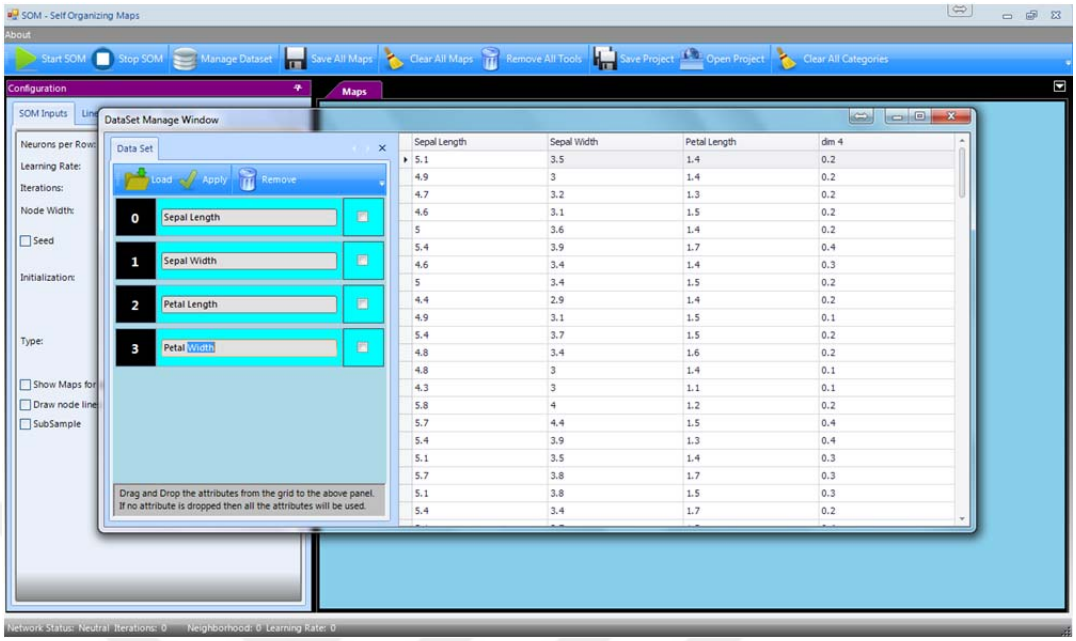


Figure 3.11. DataSet Managing Module

3.2.4.2. SOM Algorithm

To analyze and visualize input data, Self-Organizing Maps algorithm is implemented as the main algorithm of the software package. Available SOM based packages put some constraints on the values of parameters and the number of neurons, while the software of this study provides fully selectable parameters. The most important thing is that the number of neurons and their pixel size can be as big as the user decides. In other available tools (i.e. SOM Toolbox of Matlab) visualization of maps with huge number of neurons is not possible. In addition, initialization of maps can be applied using random values, random values from input data, and codevectors obtained from a K-Means training process. SOM network can also be trained using variable percentage of data from original data set, namely samples of original dataset.

All the input parameters can be written and given as input by the user in their corresponding textboxes on the software. Figure 3.12 shows the panel with all input parameters of SOM algorithm. After all the parameters are written by the user, the

training process starts by clicking on Start SOM button and the training process will start according to the given inputs from the user.

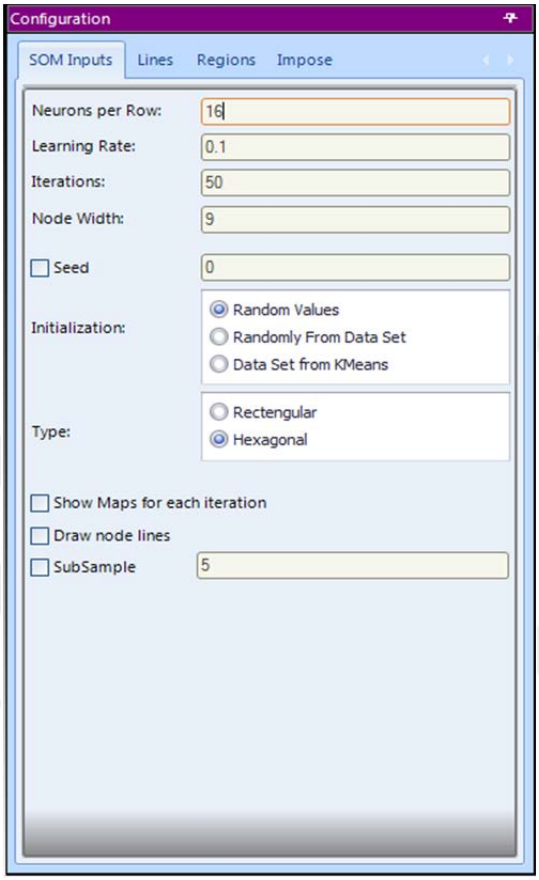


Figure 3.12. Input parameters of SOM algorithm

3.2.4.3. Visualization and Mapping

Mapping of high-dimensional input space onto a discrete two-dimensional output space (grid space) is visualized using rectangular and hexagonal grid structures. Weight values of every neuron in the map are accessible by simply clicking in the grid node (see Figure 3.13). Weight values are shown in their de-normalized state. Thereupon, the distribution of input data in the map can easily be observed. Maps can be saved as images, but in addition, the entire state of the software including input parameters, initialization selections, grid shape, maps, state of data set, and all other features can be saved in a .som file type (an xml file

containing the state of the entities of the software). Every map container has a header panel which contains buttons for saving the map as an image, copying or moving it to a new window.

Figure 3.13 and 3.14 show four maps which are constructed and trained after feeding iris data set to the network. Neurons in the first figure are visualized using hexagonal shapes while the second figure shows visualization of the maps on a rectangular grid. If the number of maps is huge a re-ordering of the maps may be necessary to make the analyzing process easier for the user. Thereupon, maps can be opened to a new window by clicking on this button, or copied to a new window by clicking button (Figure 3.14).

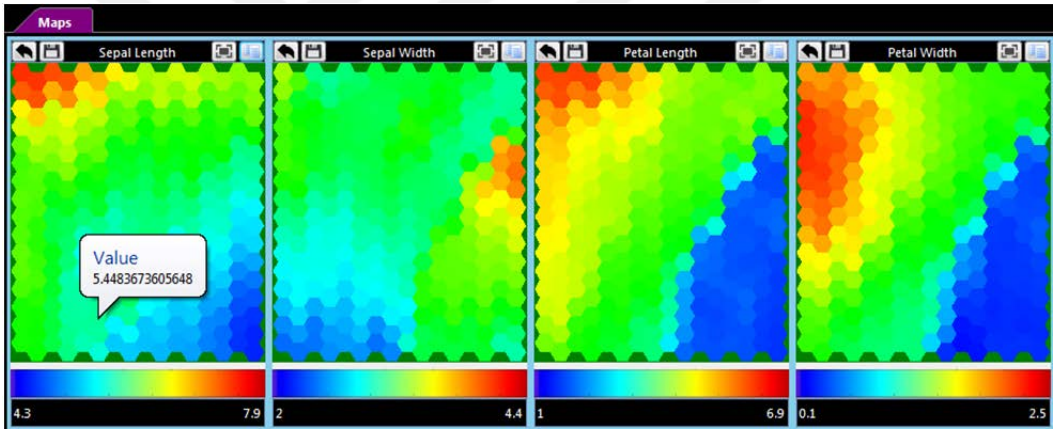


Figure 3.13. Hexagonal Maps

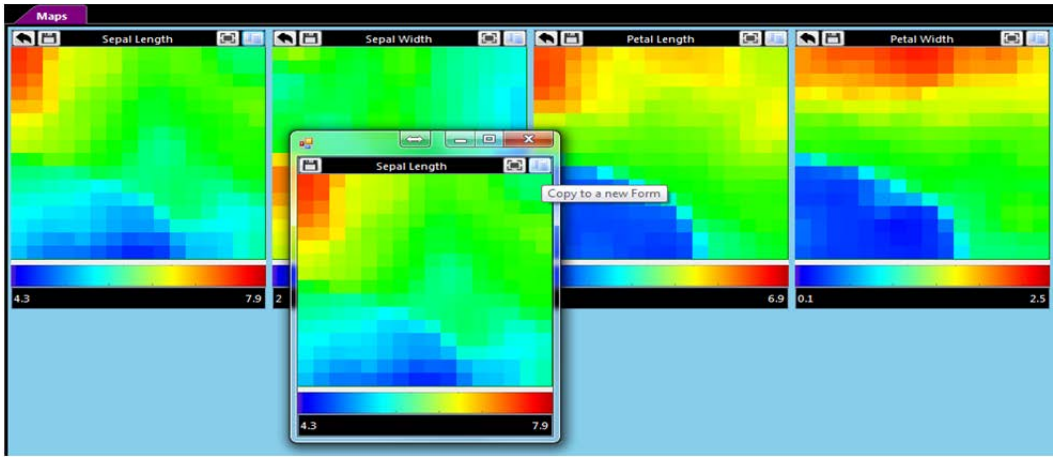


Figure 3.14. Rectangular Maps and a new form containing a copy of attribute named *Sepal Length*.

3.2.4.4. Features and Tools

3.2.4.4.(1). Lines

In self-organized maps, input data are mapped from input space according to their values. Thereby, small and big values are kept away from each other by being mapped to opposite corners of the map while average values are spread in the middle of the map, around diagonals. By clicking in the “Add Line” button under the “Lines” tab, the user can draw lines of different colors and values to these maps (Figure 3.15 and Figure 3.16). Figure 3.15 shows a window for selecting the color of the line. By adding lines the user can easily identify borders of the values in the map, thus having a better insight view of data distribution. As shown in Figure 3.16, two lines are added onto the map which of attribute *Sepal Length.* Black line defines the border of between values smaller then 5.0 and values bigger than 5.0. White line defines the border between values smaller then 6.0 and bigger then 6.0. In this way, all values between black and white lines are bigger then 5.0 and smaller then 6.0.

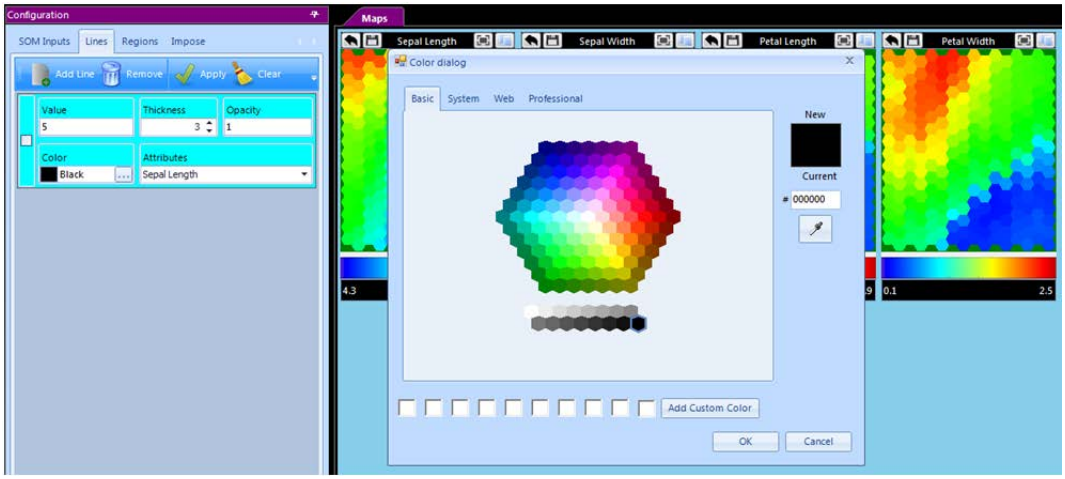


Figure 3.15**.** Drawing a line onto the map

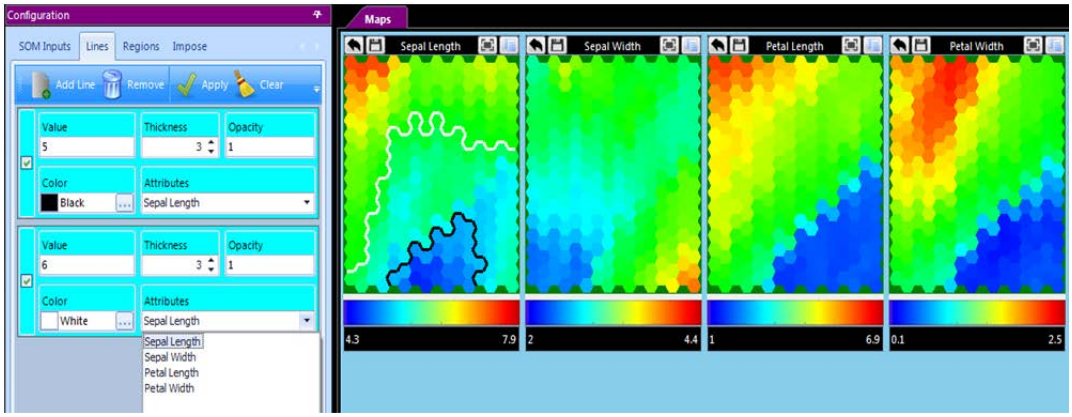


Figure 3.16. Lines (values 5.0 and 6.0) added onto the first map.

3.2.4.4.(2). Regions

To compare and analyze the correlation between data attributes, a tool for adding regions of different patterns is implemented. Similar to lines, this tool adds regions in between existing lines of the map. Regions can be of different patterns to facilitate the observation of similarity or correlation between regions of different maps. Interesting regions for the user may be drawn with the same pattern, but the way that regions are drawn is the responsibility of the user. A region can be added by right clicking the map (the menu will open) and then clicking the “Add Region” submenu (Figure 3.16). A region tool will then be added under the “Regions” tab, where the user can then select the pattern and its thickness for that region (Figure 3.18).

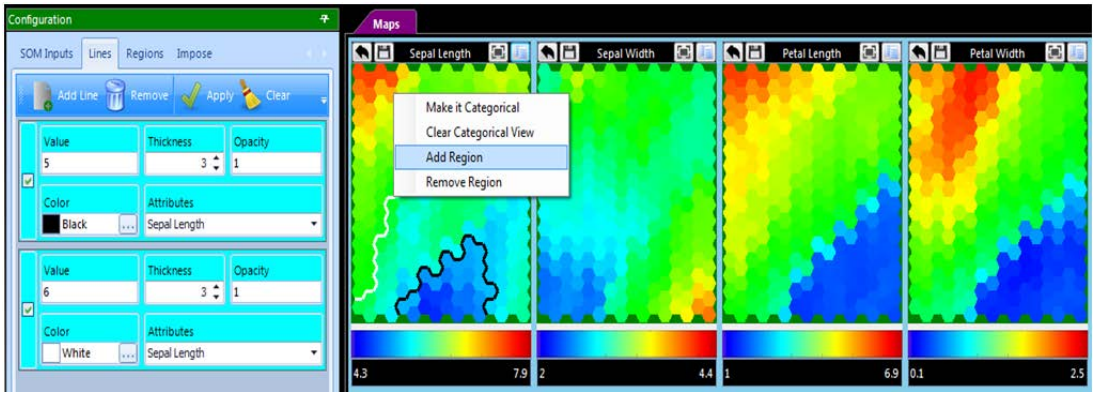


Figure 3.17. Adding a region above white line (values bigger then 6.0) to the first map

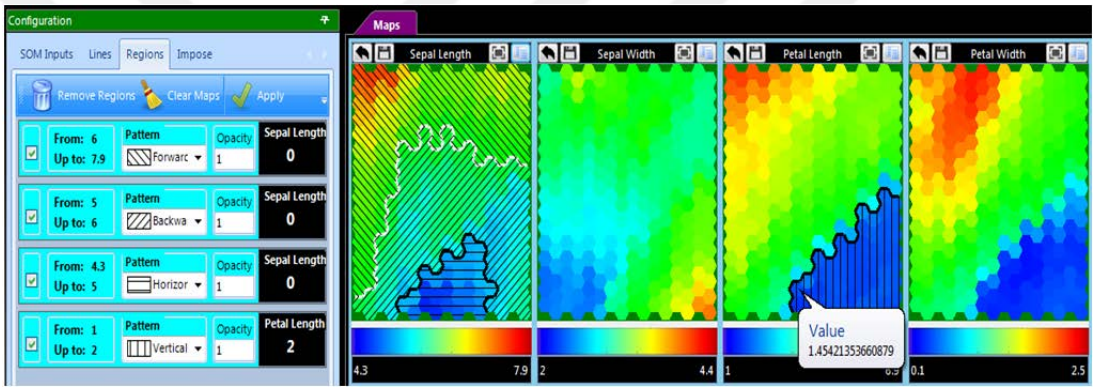


Figure 3.18. Three regions of different patterns added to the first map and another region added to the third map (smaller then 2.0).

3.2.4.4.(3). Impose

Imposing is an extension of both, the tool which adds lines and the tool which adds regions. Imposing can be used only after firstly adding lines or regions. To deeply compare maps between each other, an impose of lines from one map to another map can be applied. To add an impose control, the user should click the button “Add” under the “Impose” tab (Figure 3.19). After the control is added, the user can define from which attribute to which attribute should the impose be applied (attributes *from* and *to* can be selected on the two dropdown boxes on the impose control). The opacity of the line can be given as input from the user by writing the opacity in the corresponding textbox. Furthermore, the lines which will be imposed can be selected from the dropdown box named “Lines”. Figure 3.19 shows imposed

lines from the first map to the second map. Similar to lines, regions can also be imposed from one map to another by unchecking the “Only lines” checkbox and clicking “Apply” button (Figure 3.20). In addition, the whole map (together lines and regions) can be imposed to another map. Thus maps can be overlapped on top of each other. The opacity is selectable and varies from 0.0 up to 1.0. Imposing can be used for several purposes, depended on the user`s interest and purpose.

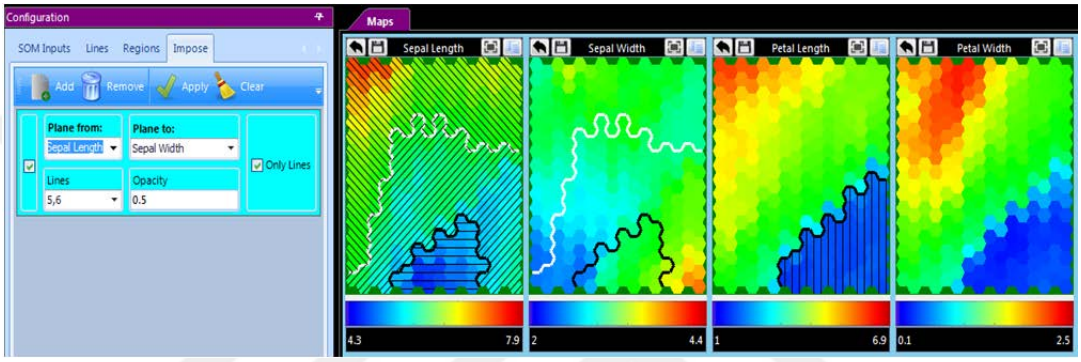


Figure 3.19. Lines of first map are imposed to the second map.

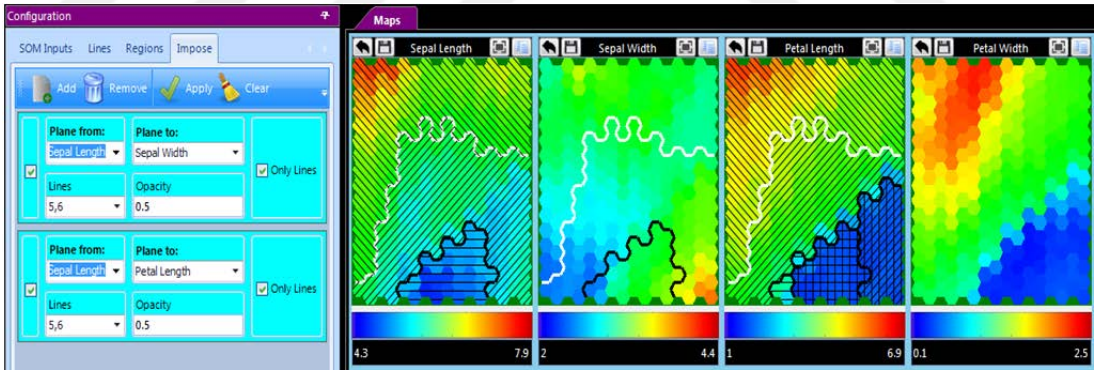


Figure 3.20. Lines of the first map are imposed onto the second map, while regions together with lines of the first map are imposed to the third map (Petal Length attribute)

3.2.4.4.(4). Categorical Regions

Sometimes, in addition to continuous values, data sets may also have categorical values. Self-Organizing Maps works fine for categorical values as well. However, it considers these values as continuous values and thus when visualizing

categorical values the corresponding map looks as it was continuous. To overcome this issue, in this study`s software user can define categorical regions, by simply colorizing it with a user-defined color.

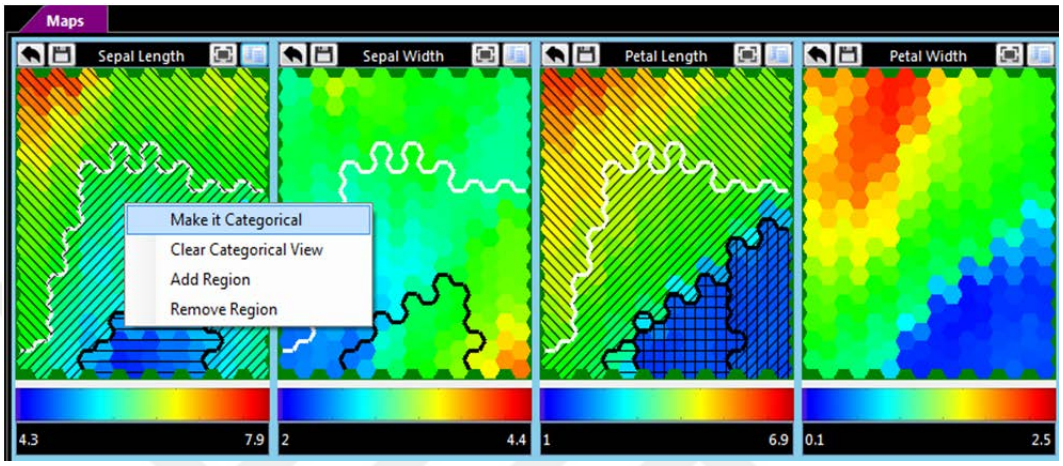


Figure 3.21. Converting the region of the first map located between values 5.0 and 6.0, to categorical.

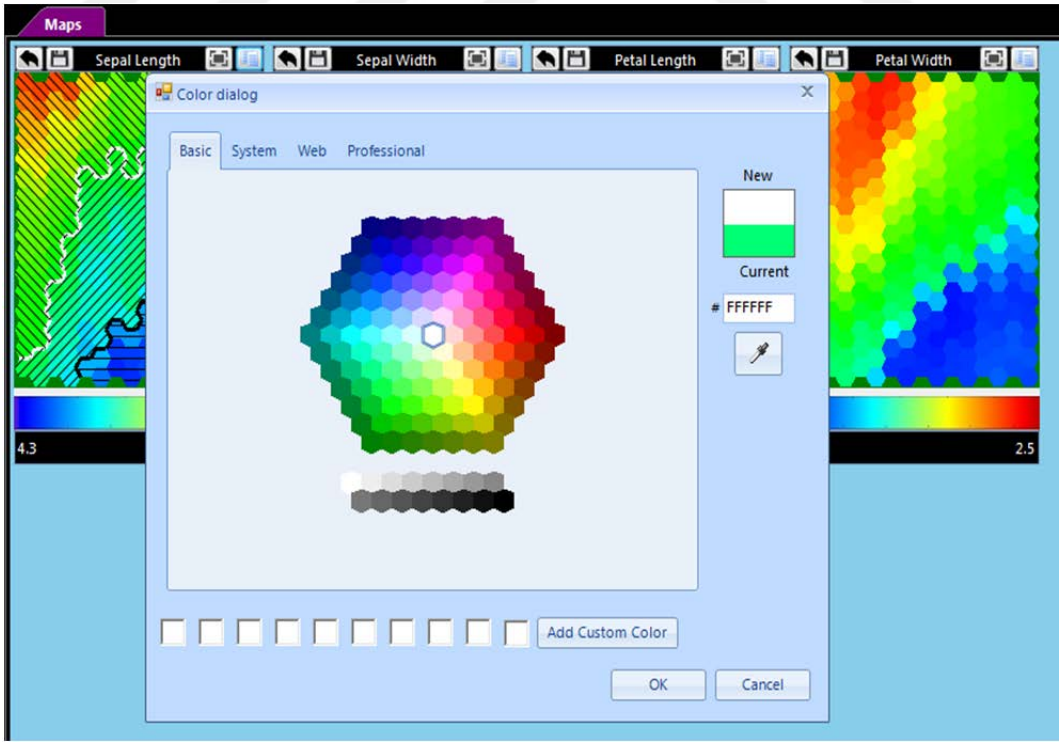


Figure 3.22**.** Selecting the color for the region, which is going to be categorical

A categorical region can be added by right clicking on the map, where the dropdown menu will be shown, and then clicking on “Make it Categorical” submenu (Figure 3.21). A window then will be opened to which will show a color map, where the user can select the desired color for that categorical region (Figure 3.22). Otherwise, if no color is selected, the color of the neuron where the user right clicked will be selected by the system. Figure 3.23 shows a categorized region of the map which represents Sepal Length attribute. In this way, because of its colors, the categorized region will not be confusing for the user when analyzing correlation between attributes.

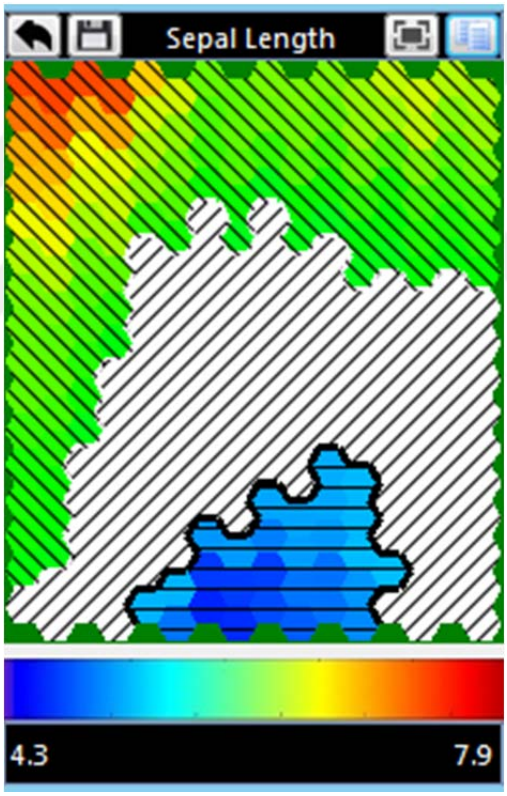


Figure 3.23. Region between white line and black line converted to categorical

4. RESULTS AND DISCCUSIONS

4.1. Performance Measurements and Testing Configurations

4.1.1. Data Selection and Testing Configurations

To observe the real performance of the algorithm for varying data set sizes, PSOM and PB-PSOM are trained with very small, average and big data sets. Furthermore, to be able to evaluate the quality of the results, all the data sets have been generated manually, using different mathematical functions for each attribute. Some of the attributes in data sets are correlated to each other. If one has prior knowledge for a data set and the correlation between attributes, then it is easier to validate results of the algorithm. Each data set has five dimensions, while the number of input patterns is determined as follows: 100, 500, 1000, 5000, 10000, 50000, and 100000 inputs. Sizes of 100, 500 and 1000 may represent small data sets. Big data sets are represented by data sets containing 50000 and 100000 inputs while the size of 5000 and 10000 may represent average data sets.

First level (first map) of PSOM is very small and it serves only for grouping input patterns into several beans (neurons). Thus, it is not logical to train a very small map with huge data sets. To roughly group a huge data set into several beans, a small sample of original data set is enough. Nevertheless, the question is how small should the subsample be? To find the suitable (optimal) size of the subsample, algorithms proposed in this study have been tested with variable sizes of samples, such as 5%, 10%, 15%, 20%, and 25%.

While performing every test case, number of iterations in SOM is fixed to 50. For PSOM and PB-PSOM, number of iterations is fixed to 50 as well, but 50 iterations for each level. Each level of PSOM performs its training process for the same number of iterations as in a full training process of SOM. This decision is an implication of our attempt to keep every level of PSOM as similar as possible to original SOM. For all test cases, learning rate starts at 0.1 and goes up to 0.01, but never smaller than 0.01 (Haykin, 1999, p. 452).

All tests of PSOM are carried out in a three level PSOM; therefore, the map in the third level has 4096 neurons. Tests are carried out in an Intel Core i5 computer with a speed of 1.8 GHz, 8 GB of RAM and a processor containing four cores. Thereupon, number of tasks in the parallel version of PSOM (PB-PSOM) for our test cases is four. Thus, data set is divided into four subgroups, where each group is given to a task of TPL (See Section 3.1.6.2). In Table 4.1 and Table 4.2 all the details of testing configuration performed in this study are shown.

Table 4.1. Configuration of Test Cases for SOM

Test Case Configuration - SOM					
#Test Case	**#Neurons**	**#Inputs**	**Starting Learning Rate**	**#Iterations**	**#Executions**
1	4096	100	0.1	50	5
2	4096	500	0.1	50	5
3	4096	1000	0.1	50	5
4	4096	5000	0.1	50	5
5	4096	10000	0.1	50	5
6	4096	50000	0.1	50	5
7	4096	100000	0.1	50	5

Table 4.2. Configuration of Test Cases for PSOM and PB-PSOM

#Test Case	#Neurons	#Inputs	Subsample Size (%)	Starting Learning Rate	#Iterations per level	#Tasks	#Executions
1	4096	100	5	0.1	50	4	5
2	4096	500	5	0.1	50	4	5
3	4096	1000	5	0.1	50	4	5
4	4096	5000	5	0.1	50	4	5
5	4096	10000	5	0.1	50	4	5
6	4096	50000	5	0.1	50	4	5
7	4096	100000	5	0.1	50	4	5
8	4096	100	10	0.1	50	4	5
9	4096	500	10	0.1	50	4	5
10	4096	1000	10	0.1	50	4	5
11	4096	5000	10	0.1	50	4	5
12	4096	10000	10	0.1	50	4	5
13	4096	50000	10	0.1	50	4	5
14	4096	100000	10	0.1	50	4	5
15	4096	100	15	0.1	50	4	5
16	4096	500	15	0.1	50	4	5
17	4096	1000	15	0.1	50	4	5
18	4096	5000	15	0.1	50	4	5
19	4096	10000	15	0.1	50	4	5
20	4096	50000	15	0.1	50	4	5
21	4096	100000	15	0.1	50	4	5
22	4096	100	20	0.1	50	4	5
23	4096	500	20	0.1	50	4	5
24	4096	1000	20	0.1	50	4	5
25	4096	5000	20	0.1	50	4	5
26	4096	10000	20	0.1	50	4	5
27	4096	50000	20	0.1	50	4	5
28	4096	100000	20	0.1	50	4	5
29	4096	100	25	0.1	50	4	5
30	4096	500	25	0.1	50	4	5
31	4096	1000	25	0.1	50	4	5
32	4096	5000	25	0.1	50	4	5
33	4096	10000	25	0.1	50	4	5
34	4096	50000	25	0.1	50	4	5

Every test case is executed 5 times. To better express the measurement values of test cases and to avoid problems of randomness, Standard Error of the Mean (SDOM) (Taylor, 1997) is calculated for every measurement. SDOM is computed using following equation:

$$SDOM = \frac{\sigma}{\sqrt{N}} \tag{4.1}$$

where σ stands for Standard Deviation of measurements and N is the number of measurements (number of test cases). In our case, N is equal to five. By assuming that measurements have Normal distribution, the twice the value of SDOM is a very common way to specify +/- error. Twice of SDOM interval has 95% probability or confidence that the true value lies within the interval, which is an accepted standard. For example, let`s say an algorithm is executed 5 times and got following numbers 5, 6, 7, 4, 5. Then the mean is 5.4 and the standard deviation is 1.1. Therefore, SDOM is $\frac{1.1}{\sqrt{5}} = 0.5$ and $2 * SDOM = 1$. Now, it can be concluded that the measurement value is 5.4 ± 1.

All tests carried out on this study, together with corresponding graphs, are supplied in APPENDIX.

4.1.2. Quantization and Topographic Errors

Several measurement methods have been used to measure the quality of Self-Organized maps. Quantization Error is a widely used measurement method. Quantization error measures the average distance between each input pattern and its closest neuron (BMU or winning neuron) in the map. In other words, quantization error calculates how fit is the self-organized map to the input patterns.

Quantization error is calculated using the following equation:

$$QE = \frac{1}{N} \| x_i - m_{x_i} \| \tag{4.2}$$

where m_{x_i} is best matching unit of input vector x_i and N is the number of input patterns. The smaller the quantization error, the closer are the input patterns to their corresponding BMU. An important feature of average quantization error is that it decreases as the map increases. This happens because if the number of neurons increases there will be more space (more neurons) to represent the data (Uriarte & Martín, 2005, pp. 19-20).

To define the topology preservation of the map is quite difficult in a discrete map (Uriarte & Martín, 2005, p. 20).

A good method to measure topology preservation of a map is the use of input patterns to determine how continues is the mapping from input space to the grid space (the map) (Uriarte & Martín, 2005, p. 20). To achieve this, topographic error is used. Topographic Error is actually the error measurement suggested by Kohonen in his book (Kohonen, 2001). Topographic error finds the percentage of all input patterns where their first and second BMUs are adjacent. The lower the topographic error is the better the map preserves the topology. Topographic error is calculated using following equation:

$$TE = \frac{1}{N} \sum_{i=1}^{N} f(x_i) \tag{4.3}$$

where N is the number of input patterns, $f(x_i)$ is a function which has the value of 1 if x_i input pattern`s first and second BMU are adjacent (neighbors) and, 0 otherwise. It is shown that topographic error increases as the number of attributes of the data increases (Uriarte & Martín, 2005, p. 20).

However, even though topographic error is used to evaluate the quality of topology preservation, it has some disadvantages and it may not be the best measurement tool to evaluate topology preservation of a map. One of the main

reasons is that topographic error does not consider diagonal neighbors in a rectangular map. All the disadvantages of this measurement method are explained in (Uriarte & Martín, 2005).

4.1.3. Time Measurements

To measure the time needed to train a network, algorithms proposed on this study are executed for every test case (see Table 4.2) and each test case is run 5 times. After execution, Standard Error of the Mean (SDOM) is computed to better express errors and their variation. Time performance for all test cases is shown in seconds. All the calculations and comparisons, are performed based on seconds as a unit of time.

To define the speed up factor of proposed methods, the following equation is used:

$$S = \frac{T_{old}}{T_{new}} \tag{4.4}$$

where T_{old} is the old execution time (i.e. unimproved version), T_{new} is the new execution time (i.e. improved version), and S is the speedup factor. In terms of execution time, this calculation means an $S\times$ speedup of newly improved version over the old version of algorithm.

4.1.4. Complexity of SOM

In terms of computations, training process of Self Organizing Maps passes through three main phases in every iteration, which can be defined as follows:

1. Computation of distances between each input pattern and all neurons in the map.

2. Comparison operations between each input pattern and all neurons in the map to select the winner neuron.
3. Weight vectors modification (update phase)

To train a map with N neurons, using a data set with M input patterns, it is necessary to compute $N \times M$ distances, $N \times M$ comparisons to determine the winner neuron and $N \times M$ weight vector updates (see Table 4.3).

Table 4.3. Number of computations required at each iteration of SOM training process

Phases	Total Number
Distances	$N \times M$
Comparisons	$N \times M$
Weights Updates	$N \times M$

4.2. Comparisons between PSOM and SOM

4.2.1. Complexity of PSOM

Unlike SOM algorithm, PSOM never works with the whole map, except with the first map (which is quite small). Instead, in every level it only processes a small number of neurons. In terms of complexity and time consumption, this is the main advantage of training process in PSOM. Like SOM, training process of PSOM passes through same phases, in every iteration and in every level (see Section 4.1.4).

In terms of these phases, the difference between PSOM and SOM is that PSOM passes through these phases in every level of the algorithm while SOM doesn`t have levels. If there are 3 levels (3 maps), training process of PSOM passes through these phases 3 times in every iteration, one time per each map. This happens because every map in every level of PSOM is actually a kind of Self-Organizing

Map. For more details, see Section 3.2.1. In addition, the complexity of first level of PSOM is the same as the complexity of SOM with the same map size.

For a PSOM with three levels and M input patterns, complexity analysis is shown in Table 4.4. In a three level PSOM, first map contains 16 neurons, while according to equation (3.14) third map contains $16^3 = 4096$ neurons. Even though, second and third map of PSOM contain a huge number of neurons, the number of calculations required to compute all distances, comparisons and weight updates of each map, is the same as the number of calculations required to compute distances, comparisons and weight updates in the first map, respectively. In other words, even though the number of levels increases, the number of computations for each level remains same as the number of computations in the first map. Thereby, for a PSOM with 3 levels, the entire number of calculations of distances, comparisons and weight updates respectively, is three times the number of calculations required to train the first map (3 x (16 x M)). (See Table 4.4).

Table 4.4. Complexity of PSOM with three levels.

Phases	**Total Number**	**Level**
Distances per level	$16\times M$	1st
	$16\times M$	2nd
	$16\times M$	3rd
Total Distances	$3\times(16\times M)$	
Comparisons per level	$16\times M$	1st
	$16\times M$	2nd
	$16\times M$	3rd
Total Comparisons	$3\times(16\times M)$	
Weight Updates per level	$16\times M$	1st
	$16\times M$	2nd
	$16\times M$	3rd
Total Weight Updates	$3\times(16\times M)$	

In a general case, where L is number of levels, K is the number of neurons in the first map and M is the number of instances, PSOM has the complexity shown in Table 4.5.

Table 4.5. Complexity of PSOM in a general case

Phases	**Total Number of Calculations**
Distances	$L \times (K \times M)$
Comparisons	$L \times (K \times M)$
Weights Updates	$L \times (K \times M)$

To train a map consisting of 4096 neurons with M input patterns, three levels of PSOM are required. According to Table 4.4 and Table 4.5, it requires the computation of $3 \times (16 \times M)$ distances, $3 \times (16 \times M)$ comparison operations to select winners and $3 \times (16 \times M)$ weight updates to modify weights vectors in one iteration. On the other hand, to train the same map size using SOM, it requires the computation of $16^3 \times M$ distances, $16^3 \times M$ comparison operations to select the winners and $16^3 \times M$ weight vector updates in one iteration. This case is shown in Table 4.6.

Table 4.6. SOM vs. PSOM for training a map consisting of 4096 neurons with M input patterns

Phases	**SOM**	**PSOM**
Distances	$16^3 \times M$	$3 \times (16 \times M)$
Comparisons	$16^3 \times M$	$3 \times (16 \times M)$
Weights Updates	$16^3 \times M$	$3 \times (16 \times M)$

The general comparison between SOM and PSOM is shown in Table 4.7.

Table 4.7. SOM vs. PSOM for training a map consisting of K^L neurons with M input patterns

Phases	SOM	PSOM
Distances	$K^L \times M$	$L \times (K \times M)$
Comparisons	$K^L \times M$	$L \times (K \times M)$
Weights Updates	$K^L \times M$	$L \times (K \times M)$

where K is the number of neurons in the first level of PSOM, L is the number of levels of PSOM and M is the number of instances.

As shown in Table 4.7, the number of calculations required in PSOM is much smaller than the number of calculations required in SOM. The number of calculations in the case of SOM is exponentially increasing as the number of levels of PSOM increases, while in the case of PSOM this increment is of linear form.

Another additional and very important feature of PSOM is that, training a PSOM network consisting of L levels provides L trained maps of different sizes. These L maps can be used for different purposes later on.

4.2.2. Quantization Error

In all test cases of this study (Table 4.1 and Table 4.2), quantization error of the maps is measured, in order to show how fit is the self-organized map to the input space. In the case of PSOM, different subsample sizes have been used in the first level. We wanted to see how the size of the subsample in the first level affects the quality of the map at the last level. Results have shown that the size of the subsample in the first level does not affect the quality of the map at the last level. Thereupon, results on this study are shown using 20% of original data set in the first level of PSOM, while the other results and graphs are given in Appendix.

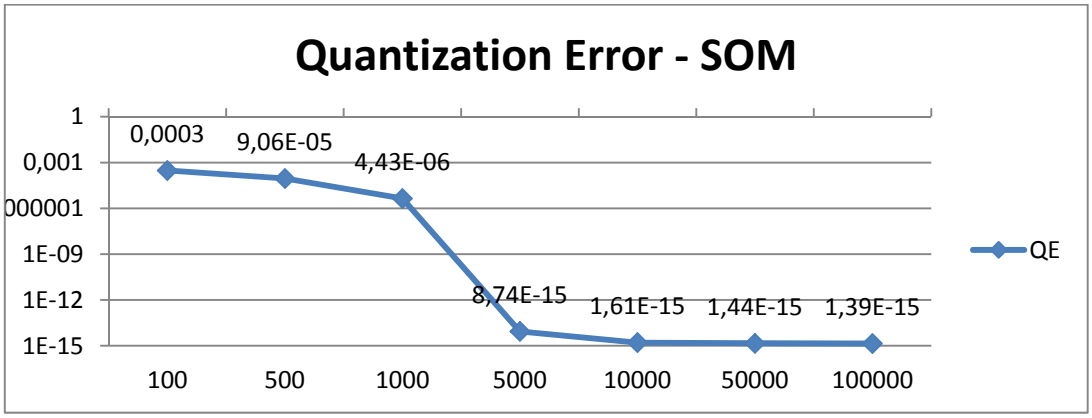
Results have shown that PSOM achieves to construct and train maps with better quantization error in all cases, for all data sets. Table 4.8 shows the SDOM (Standard Deviation of Mean) of quantization error for both, SOM and PSOM, for

seven different sizes of data sets. On the other hand, Figure 4.1, under a and b, shows plots of the mean of quantization error separately for SOM and PSOM, while the the under c the figure shows differences between SOM and PSOM.

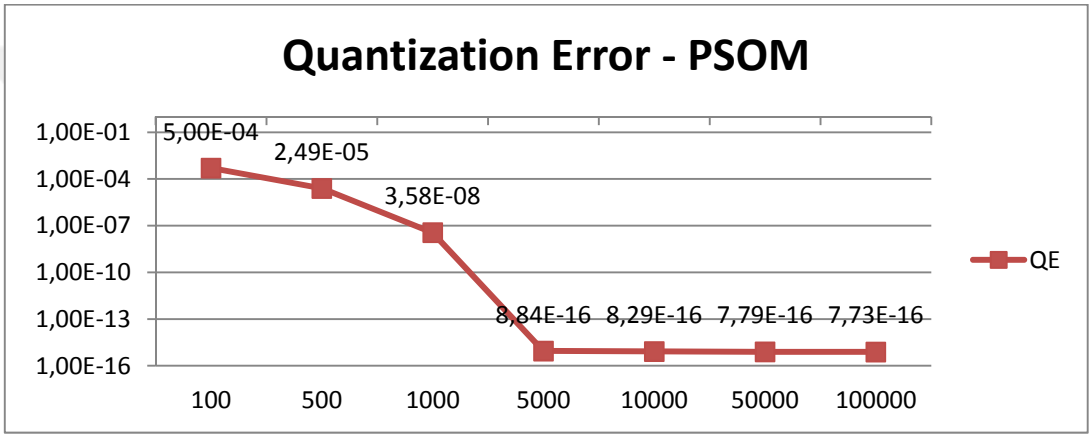
Table 4.8. Quantization Error in SOM and PSOM for different sizes of input data.

#Inputs	QE - SOM	QE - PSOM
100	0.0003 ± 2.04E-05	0.0005 ± 6.18E-05
500	9.06E-05 ± 2.53E-06	2.49E-05 ± 4.52E-06
1000	4.43E-06 ± 5.16E-07	3.58E-08 ± 2.45E-08
5000	8.74E-15 ± 4.39E-15	8.84E-16 ± 1.37E-17
10000	1.61E-15 ± 3.8E-18	8.29E-16 ± 9.75E-18
50000	1.44E-15 ± 1.69E-17	7.79E-16 ± 8.09E-18
100000	1.39E-15 ± 2.48E-21	7.73E-16 ± 1.15E-17

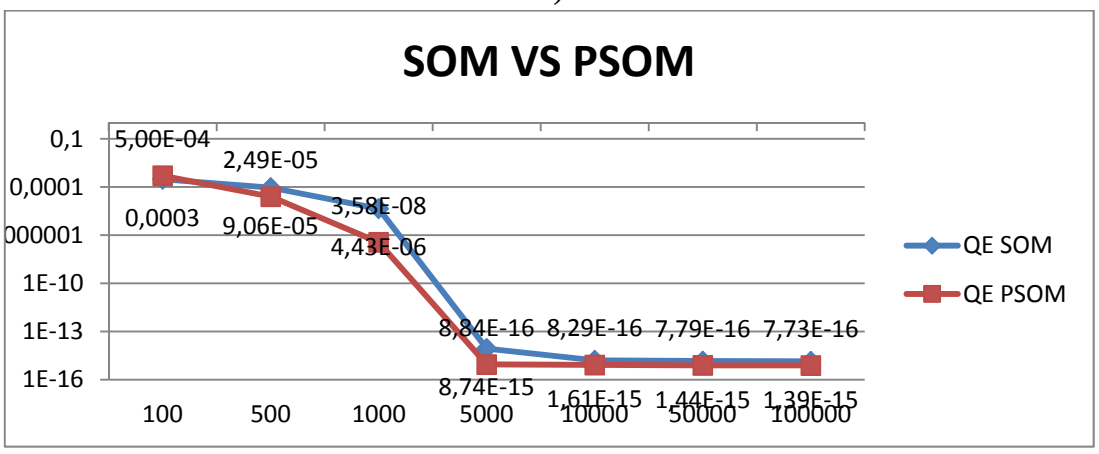
As shown in results of this section, PSOM achieves to produce maps with better quantization error. Initialization method that is implemented for second and third map, which provides a well topologically ordered map in the beginning of training process is a contributor to this quality of quantization error. The small neighborhood range is an additional contributor. Small neighborhood range allows neurons to be independently connected to their parent-neurons and furthermore, it keeps these neurons affected only by their parents and their closest neighbors, instead of being affected from neurons far away from them; thereby, the map continuously fine-tunes itself.



a)



b)



c)

Figure 4.1. Quantization Error in SOM and PSOM: a) QE in SOM, b) QE in PSOM and c) Comparisons between QE of SOM and PSOM

4.2.3. Time Measurements

Logically, expectations were that running PSOM with different subsample sizes (5%, 10%, 15%, 20% and 25%) in the first level would affect the execution time of the training process, according to the size of subsample. Thus, an execution of PSOM using 5% of original data set in the first level was expected to consume less time comparing to the time required to train PSOM using bigger subsample sizes (10%, 15%, 20% and 25%). However, results have shown that the size of subsample used in the first level, does not affect the execution time of the algorithm in small and average data sets. This happens for two main reasons; the first map is very small (only 16 neurons) and the computer where tests are performed is quite fast to be depended on the training of such a small map with such small subsamples. Even if the computer is not executing any other application on the time of execution of the algorithm, still there are some operations that operating system executes “under the hood”. In the case of small and average data sets, with subsample sizes used in our test cases (See Table 4.2), such operations executed by operating system may affect the execution time of the algorithm. This can be realized from Figure 4.2. In this figure, one can see that in small and average data sets (up to 50000 input patterns) there is not any influence of subsample size to the execution time of algorithm. As explained above, sometimes the execution time using 10% of data in the first level consumes more time than the case of 25%.

On the other hand, for large data sets (50000 inputs or more) the execution time increases as the subsample size in the first level increases. However, even in large data sets, the difference of execution time in the worst case is around 120 seconds or 2 minutes. This difference does not have any impact, and furthermore it is inconsiderable in huge data sets. For these reasons, to be consistent with comparisons of quantization error, time comparisons between SOM and PSOM are given using 20% of the data in the first level.

Table 4.9 shows the SDOM (Standard Deviation of Mean) of execution time of SOM and PSOM, using seven different sizes of input data. Speedup factor is shown for every size of input data. On the other hand, Figure 4.3 shows plots of the

mean of execution time, separately for SOM and PSOM (a and b), while the graph in c shows differences between SOM and PSOM.

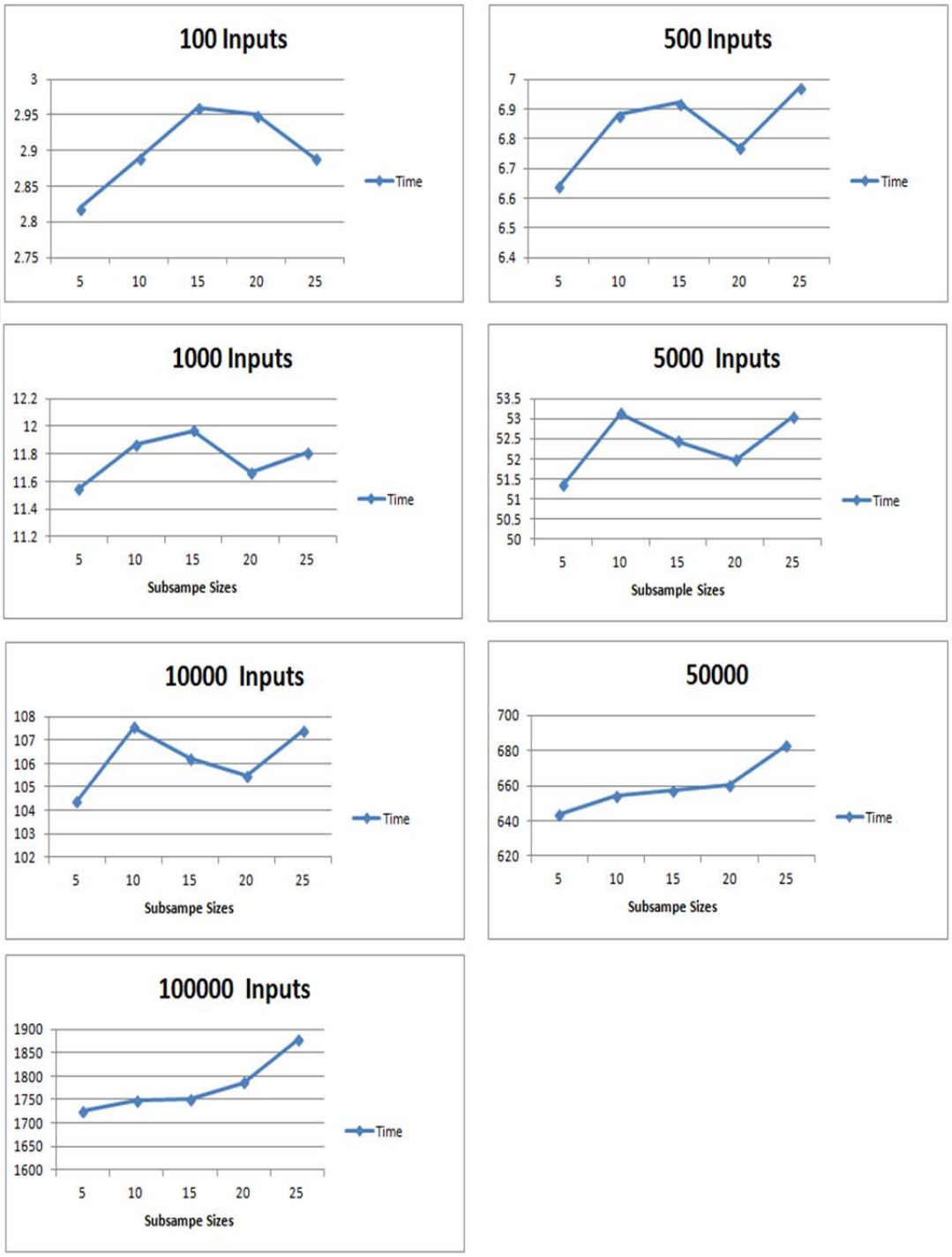
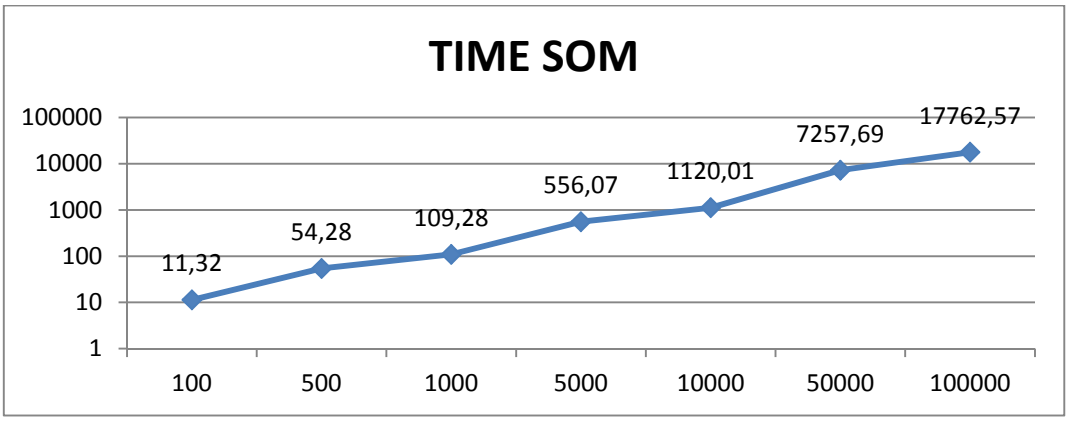
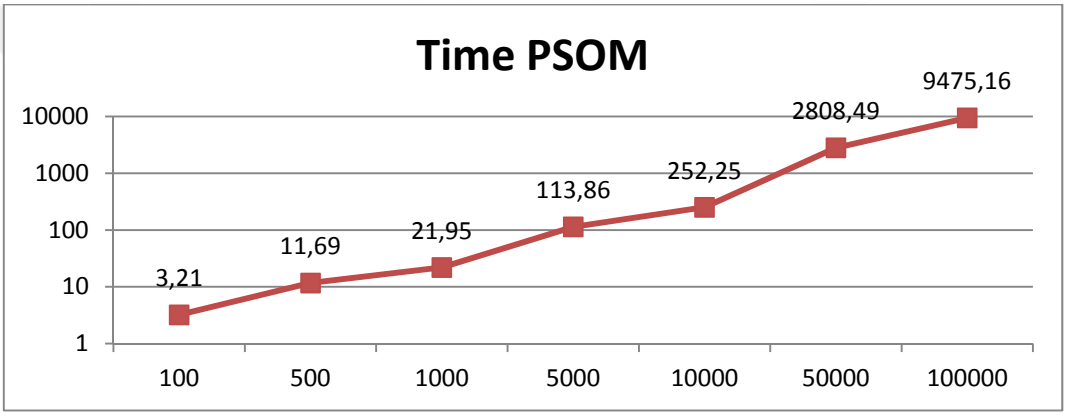


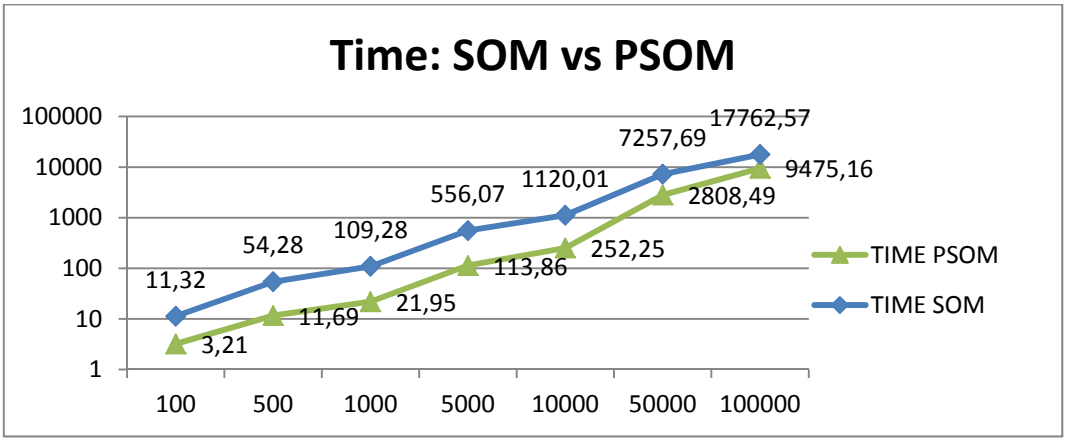
Figure 4.2. PSOM trained with different subsample size in the first level, for each data set.



a)



b)



c)

Figure 4.3. Execution Time in SOM and PSOM: a) Execution Time in SOM, b) Execution Time in PSOM and c) Comparisons between Execution Time of SOM and PSOM

Table 4.9. Execution Time of SOM and PSOM for different sizes of input data.

#Inputs	Time – SOM (seconds)	Time – PSOM(seconds)	Speedup factor
100	11.32 ± 0.15	3.21 ± 0.01	3.526479751
500	54.28 ± 0.3	11.69 ± 0.18	4.643284859
1000	109.28 ± 0.19	21.95 ± 0.2	4.978587699
5000	556.07 ± 1.37	113.86 ± 0.68	4.883804672
10000	1120.01 ± 9.5	252.25 ± 1.02	4.440079286
50000	7257.69 ± 63.54	2808.49 ± 12.1	2.58419649
100000	17762.57 ± 542.45	9475.16 ± 87.10	1.874645916

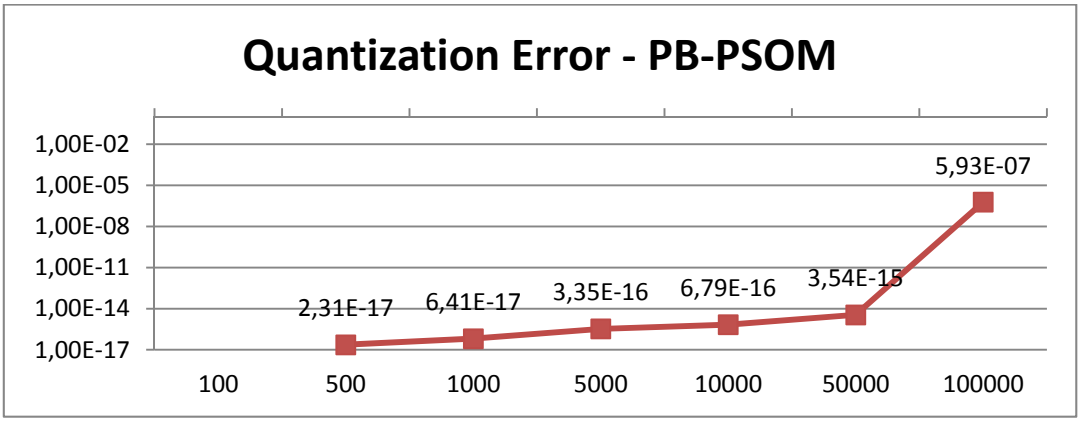
Considering results shown in Table.4.9, PSOM achieves an average speedup factor of 3.847297. In terms of execution time, this means that PSOM provides a $3.85\times$ speedup over the SOM algorithm.

4.3. Comparisons between PB-PSOM and PSOM

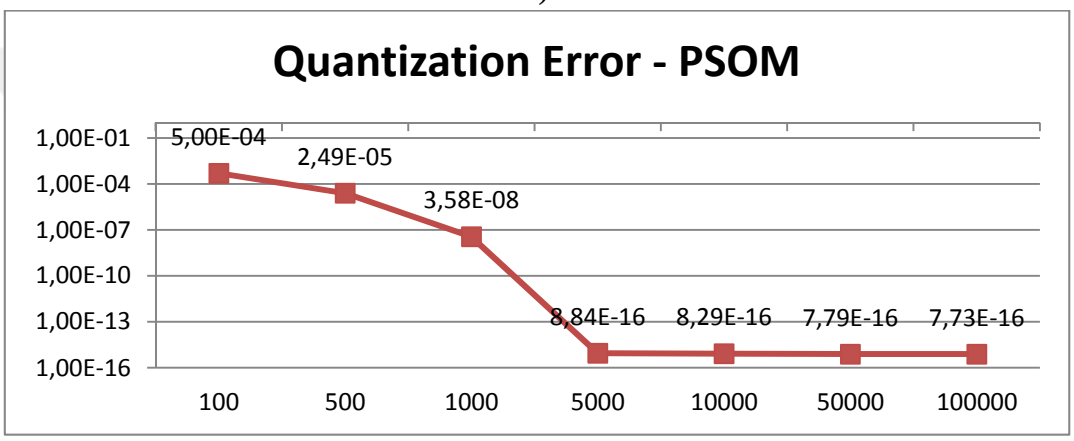
4.3.1. Quantization Error

As explained in Section 3.2.3, PSOM is further improved, in terms of time consuming, by parallelizing its Batch version (B-PSOM).

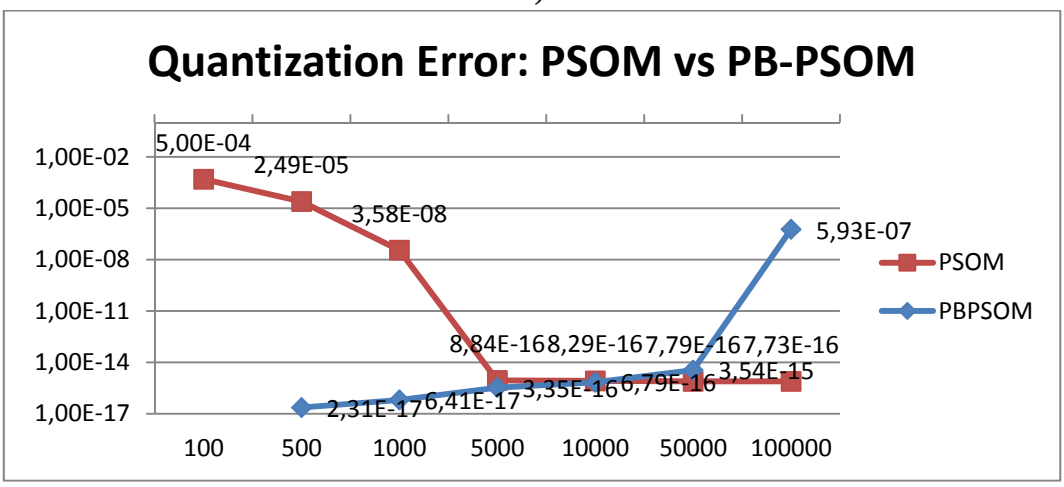
As it can be observed from Table 4.10, Parallelized Batch PSOM (PB-PSOM) achieves to construct and train maps with better quantization error for small and average data sets carried out in this study. Table 4.10 shows the SDOM (Standard Deviation of Mean) of quantization error of PB-PSOM and PSOM for seven different sizes of input data. On the other hand, Figure 4.4 shows plots of the mean of quantization error, separately for PB-PSOM and PSOM under a) and b), respectively. Differences between PB-PSOM and PSOM are shown under c).



a)



b)



c)

Figure 4.4. Quantization Error of PSOM and PB-PSOM: a) QE of PB-SOM, b) QE of PSOM and c) Comparisons between QE of PB-PSOM and PSOM

Table 4.10. Quantization Error of PB-SOM and PSOM for different sizes of input data.

#Inputs	QE - PB-SOM	QE - PSOM
100	0.00E+00	0.0005 ± 6.18E-05
500	2.31E-17 ± 2.19E-18	2.49E-05 ± 4.52E-06
1000	6.41E-17 ± 3.8E-18	3.58E-08 ± 2.45E-08
5000	3.35E-16 ± 9.6E-18	8.84E-16 ± 1.37E-17
10000	6.79E-16 ± 1.67E-17	8.29E-16 ± 9.75E-18
50000	3.54E-15 ± 8.93E-17	7.79E-16 ± 8.09E-18
100000	5.93E-07 ± 8.39E-07	7.73E-16 ± 1.15E-17

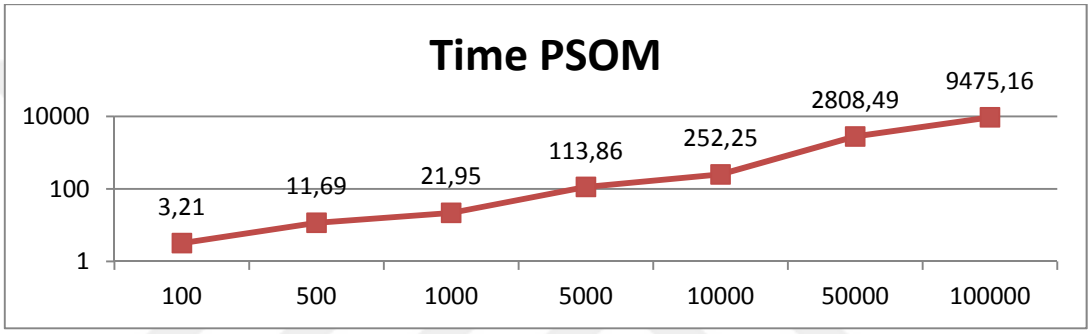
The main reason for a better performance of PB-PSOM is that the process of updating weights is not depended in the order that input patterns are presented to the network. This is actually an advantage of Batch version of SOM as well (Fort, Letremy, & Cottre, 2002; Ncker, Mrchen, & Ultsch, 2006).

For large data sets used to test the algorithm of this study, PB-PSOM has shown a higher quantization error compared to the quantization error of PSOM. This may happen because of large number of instances from the input space and the usage of another weight update equation (which is not same as in SOM or PSOM). This may be bypassed by increasing the number of neurons in the last map (starting with a bigger map in the first level), because 4096 neurons may not be enough for an appropriate mapping of every instance for a huge number data set. However, to prove this, tests need to be carried out.

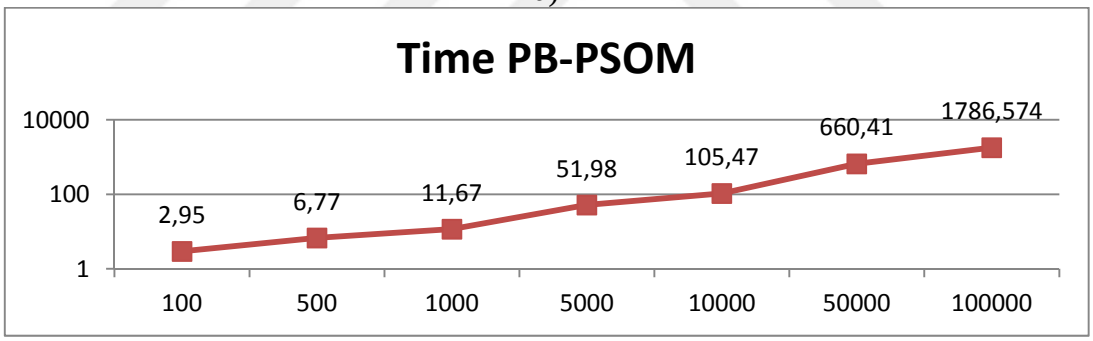
Even though, for large data sets quantization error is higher in a 3-level PSOM, considering the huge number of instances of the data set and the high speed up factor of PB-PSOM over PSOM and SOM (see section 4.3.2 and section 4.4.2) still PB-PSOM is useful for having a rough insight view of the correlation of attributes of the data set.

4.3.2. Time Measurements PB-PSOM vs PSOM

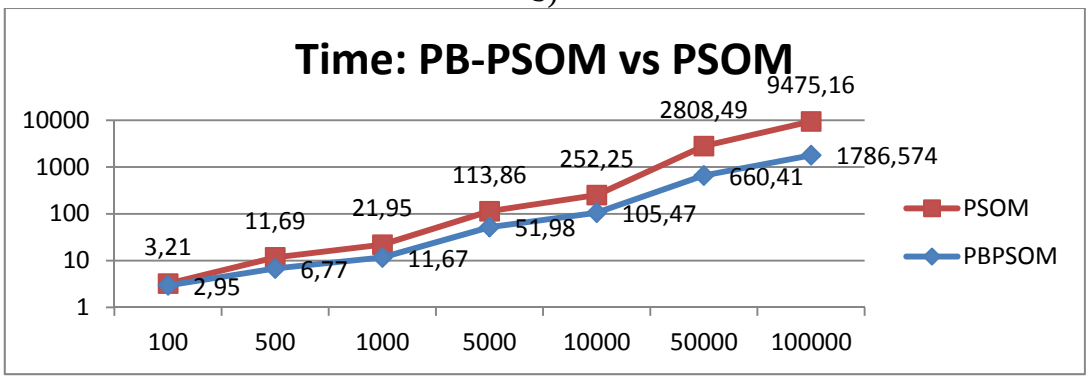
Table 4.11 shows the SDOM (Standard Deviation of Mean) of execution time of PB-PSOM and PSOM using seven different sizes of input data. Speedup factor is also shown for every size of input data. On the other hand, Figure 4.5 shows plots of the mean of execution time of PB-PSOM and PSOM under a) and b), respectively. The graph in c) shows differences between PB-PSOM and PSOM.



a)



b)



c)

Figure 4.5. Execution Time of PB-PSOM and PSOM: a) Execution Time of PSOM, b) Execution Time of PB-PSOM and c) Comparisons between Execution Time of PB-PSOM and PSOM

Table 4.11. Execution Time and speedup factor of PB-PSOM and PSOM, for different sizes of input data.

#Inputs	Time - PSOM	Time - PB-PSOM	Speedup factor
100	3.21 + - 0.01	2.95 + - 0.06	1.088135593
500	11.69 + - 0.18	6.77 + - 0.15	1.726735598
1000	21.95 + - 0.2	11.67 + - 0.04	1.880891174
5000	113.86 + - 0.68	51.98 + - 0.14	2.190457868
10000	252.25 + - 1.02	105.47 + - 0.81	2.391675358
50000	2808.49 + - 12.1	660.41 + - 0.49	4.252646083
100000	9475.16 + - 87.10	1786.574 + - 9.71	5.303536266

Considering results shown in Table 4.11, the average speedup factor is calculated for all sizes of input data. By analyzing the speedup factor in Table 4.11, it can easily be observed that the speedup factor of PB-PSOM algorithm over PSOM algorithm increases as the number of input data increases. Based on this it can be concluded that PB-PSOM is much faster than PSOM when processing large data sets. This is normal, because parallelization techniques perform better in large data sets when inter-core (inter-task) communication rarely occurs. Parallelization techniques should only be applied in high amount of workloads, otherwise, instead of speeding it up, the parallelization may make the system even slower. This happens because of the time consumed while performing many inter-processor communications, since in small workloads the inter-communication between tasks occurs more often than in high workloads.

Considering these constraints, it can be concluded that PB-PSOM achieves an average speed up factor of 2.69 over PSOM, for all kind of data sets. However, in general, parallelization techniques that are based on data partitioning approach are employed to process huge data sets. Thereupon, the speed up factor of PB-PSOM in large data sets is 4.78. In terms of execution time, this means a $4.78\times$ speedup of PB-PSOM over PSOM algorithm.

The highest number of instances in tests of this study is 100000 and the average speedup factor is measured only for data sets shown in Table 4.11.

However, in Figure 4.5, under c), the lines showing execution time of PB-PSOM and PSOM are going far away from each other as the size of data sets increases. This occurrence can also be realized in Figure 4.6 where speedup factor of PB-PSOM over PSOM ever increases as the input size increases. This leads to an even higher average of speedup factor (higher then 4.78), as the size of data sets increases.

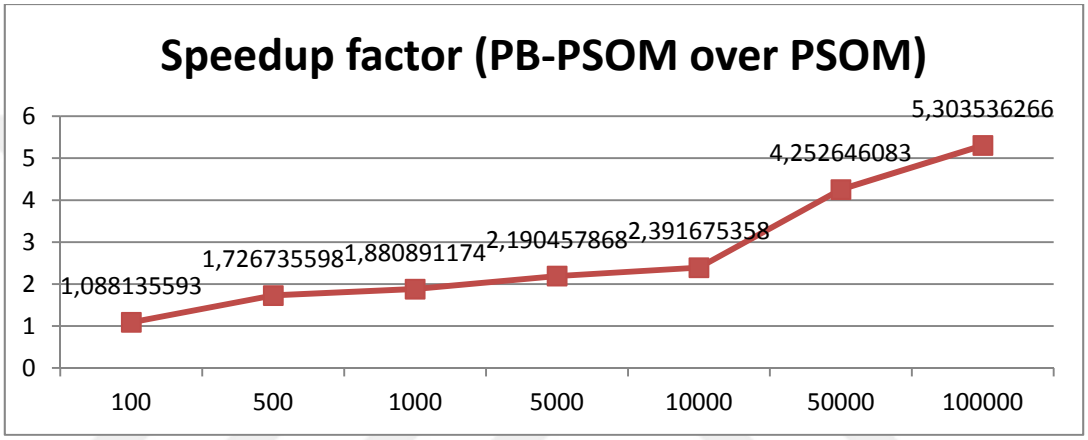


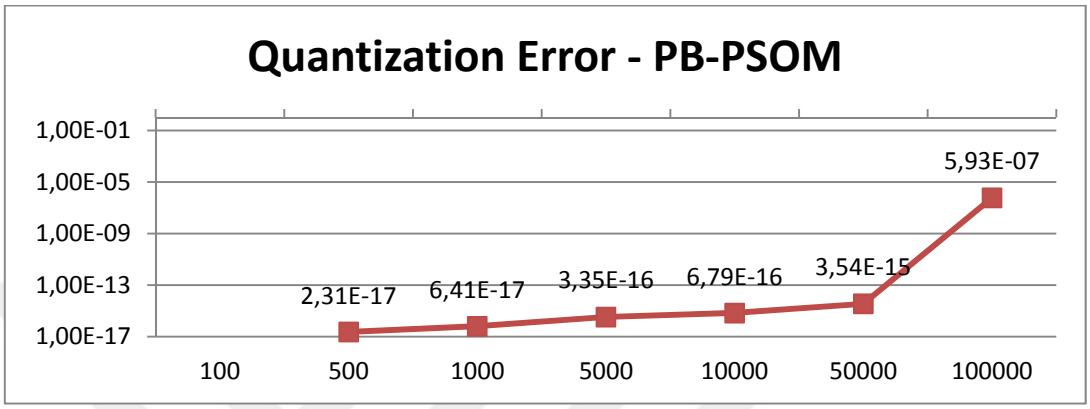
Figure 4.6. Increment trend for speedup factor of PB-PSOM over PSOM, as the size of input data increases.

4.4. Comparisons Between PB-PSOM and SOM

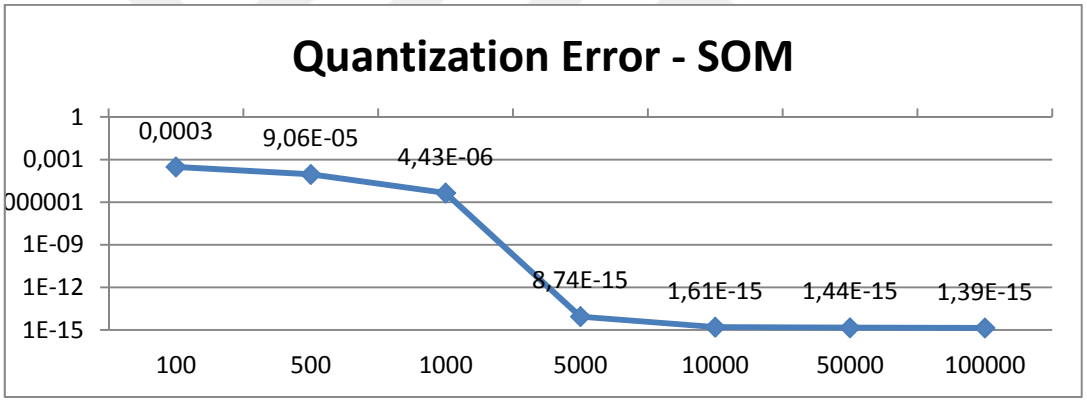
4.4.1. Quantization Error

As it can be observed from Table 4.12, PB-PSOM achieves to construct and train maps with better quantization error for small and average data sets carried out in this study. Table 4.12 shows the SDOM (Standard Deviation of Mean) of quantization error of PB-PSOM and SOM for seven different sizes of input data. On the other hand, Figure 4.7 shows plots of the mean of quantization error, separately for PB-PSOM and SOM under a) and b), respectively. Differences between PB-PSOM and SOM are shown under c).

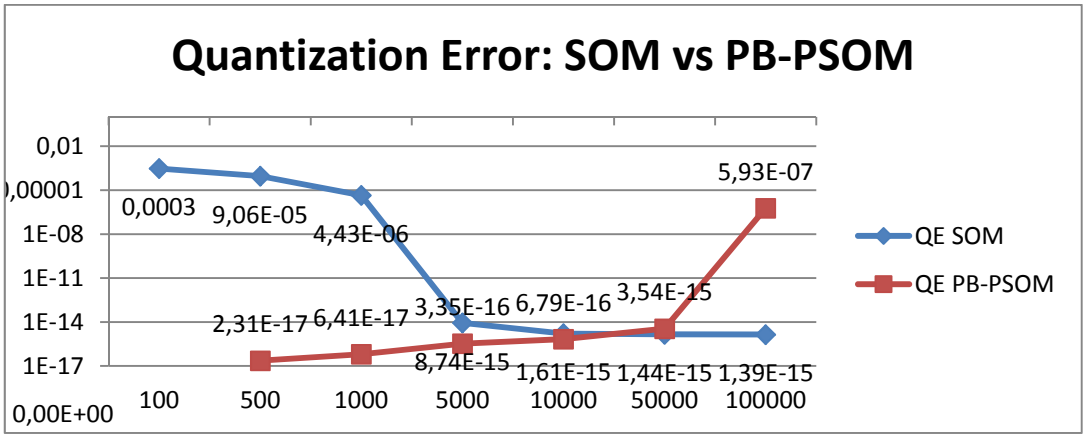
The discussion about the fact that the quantization error of PB-PSOM over SOM is lower on small and average data sets and a higher on large data sets are the same as in section 4.3.1.



a)



b)



c)

Figure 4.7. Quantization Error of PSOM and PB-PSOM and comparisons between them

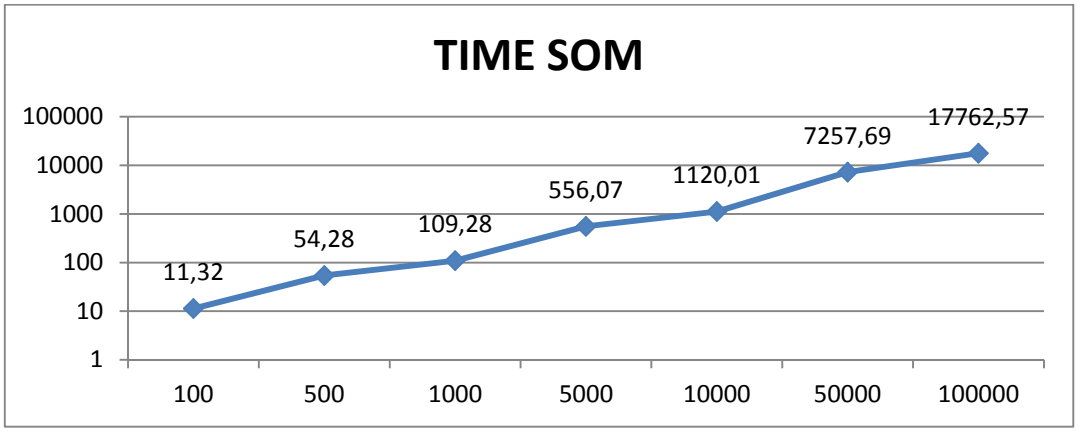
Table 4.12: Quantization Error of PB-PSOM and SOM for different sizes of input data.

#Inputs	QE - PB-PSOM	QE - SOM
100	0.00E+00	0.0003
500	2.31E-17 ± 2.19E-18	0.0000906
1000	6.41E-17 ± 3.8E-18	0.00000443
5000	3.35E-16 ± 9.6E-18	8.74E-15
10000	6.79E-16 ± 1.67E-17	1.61E-15
50000	3.54E-15 ± 8.93E-17	1.44E-15
100000	5.93E-07 ± 8.39E-07	1.39E-15

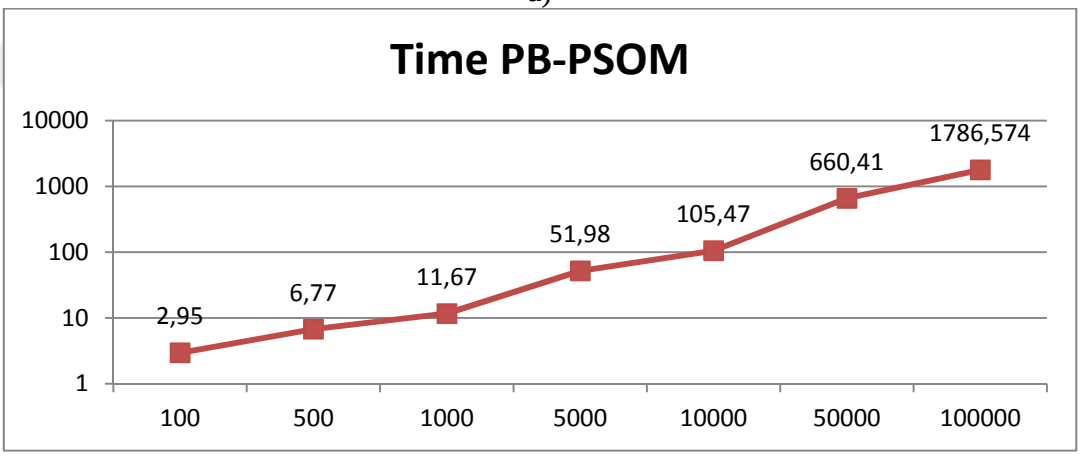
4.4.2. Time Measurements PB-PSOM vs SOM

Results have shown that PB-PSOM outperforms SOM algorithm, in terms of time consuming. Similar to Section 4.3.2, Table 4.13 lists execution time and speed up factor for every size of input data. Furthermore, Figure 4.8 plots execution time of SOM and PB-PSOM. The behavior of PB-PSOM toward SOM is same as toward PSOM. This means that the speedup factor increases as the number of input data increases. Applying same logic (as in Section 4.3.2), the average speedup factor of PB-PSOM over SOM for all data sets tested is 9.07, while the average speedup of PB-PSOM over SOM in large data sets is 10.46.

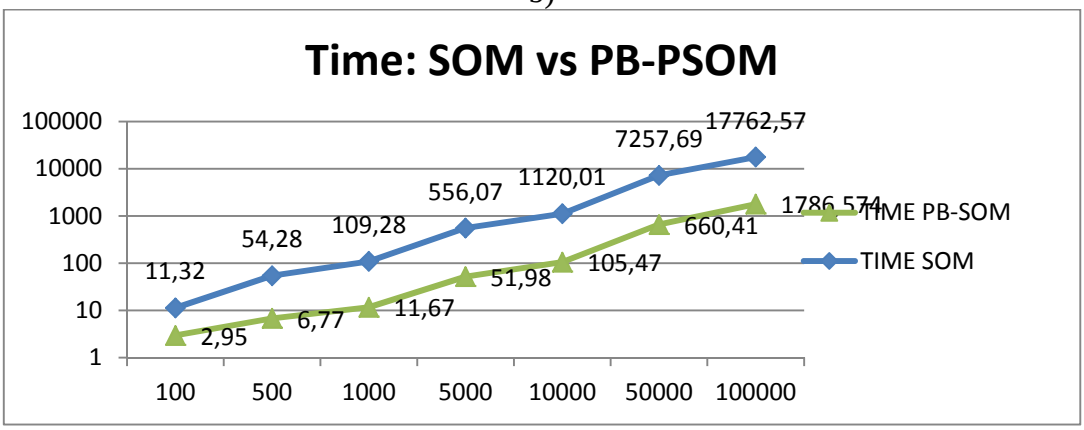
In terms of execution time, PB-PSOM provides a 10.46x speedup over the SOM algorithm.



a)



b)



c)

Figure 4.8. Execution Time of PB-PSOM and SOM: a) Execution Time of SOM, b) Execution Time of PB-PSOM and c) Comparisons between Execution Time of PB-PSOM and SOM

Table 4.13. Execution Time and speedup factor of PB-PSOM and SOM, for different sizes of input data.

#Inputs	Time - SOM	Time - PB-PSOM	Speedup factor
100	11.32 + - 0.15	2.95 + - 0.06	3.837288136
500	54.28 + - 0.3	6.77 + - 0.15	8.017725258
1000	109.28 + - 0.19	11.67 + - 0.04	9.364181662
5000	556.07 + - 1.37	51.98 + - 0.14	10.69776837
10000	1120.01 + - 9.5	105.47 + - 0.81	10.61922822
50000	7257.69 + - 63.54	660.41 + - 0.49	10.98967308
100000	17762.57 + - 542.45	1786.574 + - 9.71	9.942252602

5. CONCLUSIONS AND FUTURE WORK

The first contribution of this study is a new hierarchy-based (pyramid) approach that is capable of processing and visualizing huge data sets by mapping them from higher dimensional spaces to a two dimensional discrete space and furthermore, it facilitates the process of analyzing data sets. The main important feature of newly developed algorithm is that the process of mapping consumed less amount of time and produced results of a better quality comparing to other ANN algorithms. Actually, the only ANN algorithm that successfully achieves to incorporate all three processes, facilitation of analysis of data sets, mapping and visualizing together in one tool, is Self-Organizing Maps. Comparing to SOM, the method developed in this study (PSOM), outperforms it in terms of computational cost, time consumption, input space fitting in the discrete two-dimensional space and topology preservation.

The current study has further improved PSOM by taking advantage of multi-core environments and parallelization techniques. Thereby, a parallelized Batch version of PSOM (PB-PSOM) is implemented by partitioning the input space onto several cores of processor (data partition technique). This method achieves to accomplish its workload by consuming even less amount of time than PSOM. In addition, PB-PSOM outperforms PSOM and SOM in terms of quality of the results in small and average data sets.

The last, but not the least contribution of this study is the design and development of a software package, which implements both, PSOM, PB-PSOM and SOM, and has the ability of facilitation of analysis and visualizing data sets in an interaction with the user, by involving extra features of data analysis techniques.

To evaluate the performance of the proposed contributions, a series of experiments (210 test cases) are carried out on seven different sizes of input data. The evaluation process has carefully involved comparisons of results obtained from this study`s outcome with the results obtained from SOM algorithm. Experiments have shown significant improvements in terms of speedup, input space fitting onto the maps, and computational cost.

Methods proposed in this study can be implemented in all the fields that require any kind of data analysis, data classification, data clustering, data visualization and data mapping from high dimensional spaces to lower (typically two) dimensional discrete space. Furthermore, the software package developed on this study could be useful for analyzing data sets interactively with the user.

However, even though suggested methods outperform SOM algorithm, still there may be a necessity to experiment it with other kind of data sets such as spatial data sets and document collections, or in some kind of classification problems. Regarding our future perspective, we plan to further test these methods on some other data sets and classification problems.

REFERENCES

LO, Z. P., YU, Y., & BAVARIAN, B. (1993). Analysis of the convergence properties of topology preserving neural networks. *IEEE Transactions on Neural Netw()rks*, 207-220.

CECCARELLI, M., PETROSINO, A, & VACCARO, R. (1993). Competitive neural networks on message-passing parallel computers. *Concurrency: Practice and Experience*, 449-470.

DUCH, W., & A.NAUD. (1996). Multidimensional Scaling and Self-Organizing Maps. *International Conference on Physics Computing*, 1-3.

ERWIN, E., OBERMAYER, K., & SCHULTEN, K. (1992). Self-organizing maps: Stationary states, metastability and convergence rate. *Biological Cybernetics*, 35-35.

FORT, J. C., LETREMY, P., & COTTRE, M. (2002). Advantages and drawbacks of the batch Kohonen algorithm. *10th-European-Symposium on Artifcial Neural Networks* (pp. 223-230). Bruges: Esann.

GERSHO, A., & GRAY, R. M. (1991, November 30). *Vector Quantization and Signal Compression.* Boston/Dordrecht/London: KLUWER ACADEMIC PUBLISHERS.

HAYKIN, S. (1999). *Neural Networks, A comprehensive Foundation* (2nd ed.). Hamilton, Ontario, Canada: Prentice Hall International, Inc.

HE, B., FANG, W., GOVINDARAJU, N. K., & LUO, Q. (2008). A Map Reduce Frame-work on Graphics Processors. *17th International Conference on Parallel Architectures and Compilation Techniques* (pp. 260-269). San Francisco, CA, USA: ACM New York, NY, USA ©2008.

IENNE, P., THIRAN, P., & VASSILAS, N. (1997). Modified self-organizing feature map algorithm for efficient digital hardware implementation. *IEEE Transactions on Neural Networks*, 315-330.

TAYLOR. J.R. (1997). *An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements* (2nd ed.). Suasalito, California: University Science Books.

KELLEY, & JOHN, L. (1975). *General topology.* New York: Springer-Verlag.

KOHONEN, T. (1993). Things you haven't heard about the Self-Organizing Map. , Piscataway, NJ, IEEE, IEEE Service Center. *ICNN'93, International Conference on Neural Networks* (pp. 1147 - 1156). San Francisco, CA: IEEE.

KOHONEN, T. (2001). *Self-Organizing Maps* (3rd ed.). Berlin, Heidelberg, New York: Springer.

KOHONEN, T., KASKI, S., LAGUS, K., & SALOJÄRVI, J. (2000, MAY). Self-Organization of a Massive Document Collection. *IEEE TRANSACTIONS ON NEURAL NETWORKS, 11*, 574-585.

LAWRENCE, R. D., ALMASI, G. S., & RUSHMEIER, H. E. (1999). A Scalable Parallel Algorithm for Self-Organizing Maps with Applications to Sparse Data Mining Problems. *Data Mining and Knowledge Discovery, 3*, 171–195.

LEEUW, J. D., & HEISE, W. (1982). Theory of multidimensional scaling. (P. R. Krishnaiah, & L. N., Eds.) *Handbook of statistics, 2*, 285–316.

LINDE, Y., BUZO, A., & GRAY, R. M. (1980). An Algorithm for Vector Quantizer Design. *Communications, IEEE Transactions, 28*(1), 84-95.

MANN, R., & HAYKIN, S. (1990). A parallel implementation of Kohonen feature maps on the Warp systolic computer. *Proc. Int. Joint Conf. Neural Networks*, 84-87.

MEIJERING, ERIK. (2002). A chronology of interpolation: from ancient astronomy to modern signal and image processing, 90 (3):. *Proceedings of the IEEE*, 319–342.

MYKLEBUST, G., & SOLHEIM, J. G. (1995). Parallel self-organizing maps for actual applications. *Proceedings of the IEEE International Conference on Neural Networks*, 1054 - 1059.

NCKER, M., MRCHEN, F., & Ultsch, A. (2006). An algorithm for fast and reliable ESOM learning. *14th European Symposium on Artificial Neural Networks* (pp. 131-136). Bruges, Belgium: ESANN.

OPENSHAW, S., & TURTON, I. (1996). A parallel kohonen algorithm for the classification of large spatial datasets. *Computers and GeoSciences, 22*(9), 1019-1026.

PIKE, R., DORWARD, S., GRIESEMER, R., & QUINLAN, S. (2003). Interpreting the Data: Parallel Analysis with Sawzall. *Scientific Programming Journal*, 227-298.

REINDERS, J. (2007, September 05). *Rules for Parallel Programming for Multicore*. Retrieved from DrDobbs: http://www.drdobbs.com/parallel/rules-for-parallel-programming-for-multi/201804248

SALEEM, S., IKRAMULLAH LALI, M., & NAWAZ, M. S. (2014). Multi-Core Program Optimization: Parallel Quick Sort in Intel Cilk Plus. *1st International Conference on Modern Communication & Computing Technologies* (p. 1). Sindh: QUEST.

SAMMON, W. J. (1969). A nonlinear mapping for data structure analysis. *IEEE Trans. Comput, C-18*, 401–409.

SILVA, B., & MARQUES, N. (2007). A Hybrid Parallel SOM Algorithm for Large Maps in Data-Mining. *Proceedings of the Portuguese Conference on Artificial Intelligence*, 6-10.

TACHIBANA, K., & FURUHASHI, T. (2007). Self-Organizing Map with Generating and Moving Neurons in Visible Space. *Journal of Advanced Computational Intelligence and Intelligent Informatics, 11*, 626-632.

TACHIBANA, K., SUGIMOTO, N., & SHIOGAMA, H. (2009). Visualization of Huge Climate Data with High-Speed Spherical Self-Organizing Map. *Journal of Advanced Computational Intelligence and Intelligent Informatics, 13*, 210-216.

TORGERSON, S. W. (1952). Multidimensional scaling: I. Theory and method, Psychometrica. *Psychometrika, 17*, 401-419.

URIARTE, E. A., & DÍAZ MARTÍN, F. (2005). Topology Preservation in SOM. *International Journal of Applied Mathematics and Computer Sciences*, 19-22.

BIOGRAPHY

I am Drilon Jahiri, born on 07.04.1988 in the capital city of Kosova, Prishtina. I finished my primary and secondary school in Prishtina. On 2008, I started my bachelor studies in the direction of Computer Sciences in the department of Mathematics,in University of Prishtina. In 2010, I finished my bachelor studies and I immediately started working as a software developer in a company in Prishina. One year later, on 2012, I started my Master Studies in the department of Computer Engineering in Cukurova University, in Adana, as a scholarship holder of Turkey Government scholarship.

APPENDIX

Appendix 1. Test results and SDOM values for SOM Algorithm

Map and Data Set Information				SOM			
Nodes	**Nr. Inputs**	**Dim**	**Subsampled**	Iterations	**Time**	**QE**	**TE**
4096	100	5	5	50	11.32 + - 0.15	0.0003 + - 2.04E-05	0.48 + - 0.04
4096	500	5	5	50	54.28 + - 0.3	9.06E-05 + - 2.53E-06	0.25 + - 0.04
4096	1000	5	5	50	109.28 + - 0.19	4.43E-06 + - 5.16E-07	0.097 + - 0.04
4096	5000	5	5	50	556.07 + - 1.37	8.74E-15 + - 4.39E-15	0
4096	10000	5	5	50	1120.01 + - 9.5	1.61E-15 + - 3.8E-18	0.07 + - 0.005
4096	50000	5	5	50	7257.69 + - 63.54	1.44E-15 + - 1.69E-17	0.36 + - 0.04
4096	100000	5	5	50	17762.57 + - 542.45	1.39E-15 + - 2.48E-21	0.52 + - 0.12

Appendix 2. Test results and SDOM values for PSOM Algorithm

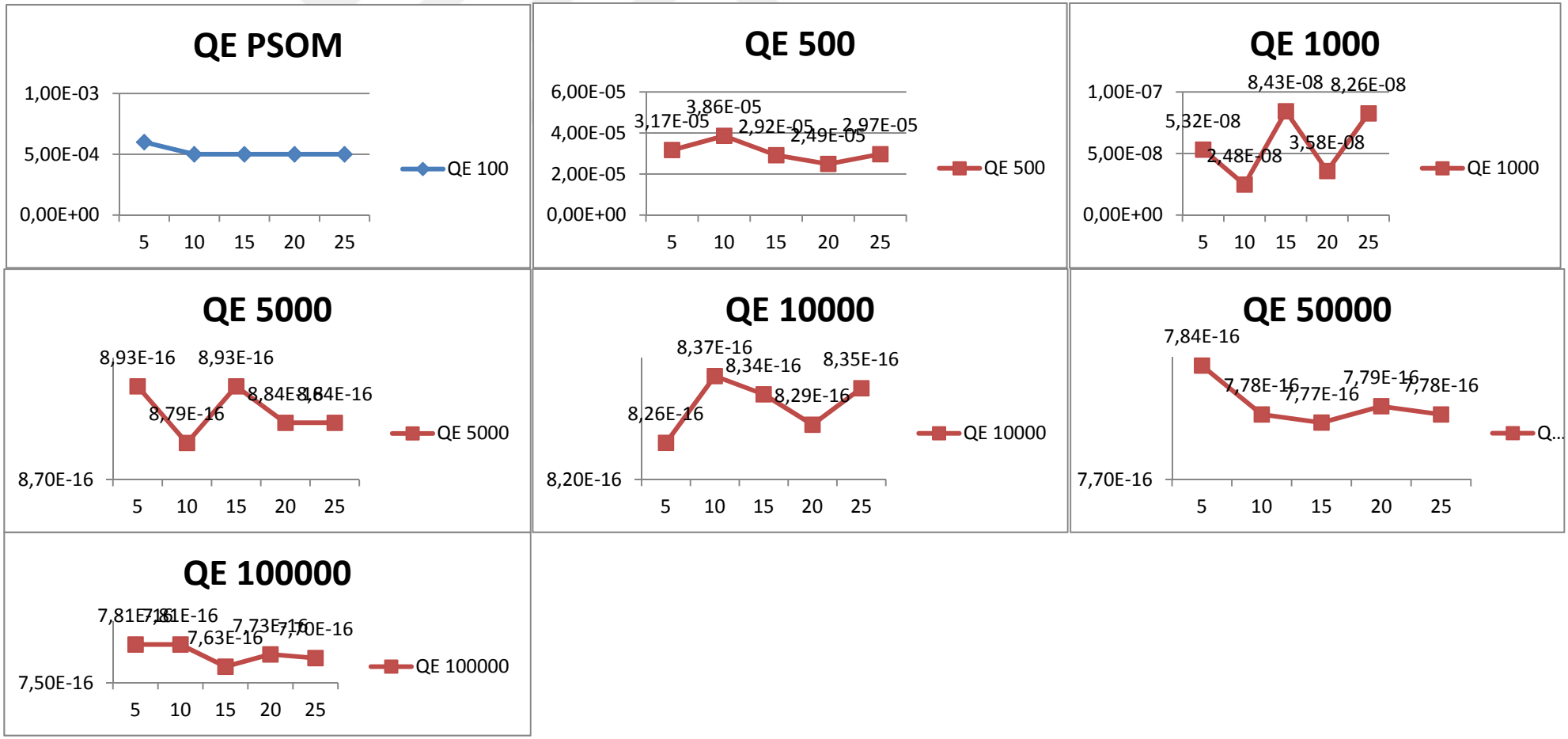
Map and Data Set Information				PSOM					
#Nodes	#Inputs	#Dim	Subsampled (%)	Iterat each Layer	Nodes in 1st Layer	Nr. Layers	Time	QE	TE
4096	100	5	5	50	16	3	4.16 + - 0.03	0.0006 + - 8.78E-05	0.206 + - 0.05
4096	500	5	5	50	16	3	14.91 + - 0.96	3.17E-05 + - 6.7E-06	0.04 + - 0.007
4096	1000	5	5	50	16	3	22.15 + - 0.4	5.32E-08 + - 3.04E-08	0.03 + - 0.02
4096	5000	5	5	50	16	3	115.67 + - 0.36	8.93E-16 + - 1.07E-17	0.17 + - 0.02
4096	10000	5	5	50	16	3	256.73 + - 1.29	8.26E-16 + - 1.22E-17	0.18 + - 0.02
4096	50000	5	5	50	16	3	2817.20 + - 24.63	7.84E-16 + - 5.9E-18	0.55 + - 0.03
4096	100000	5	5	50	16	3	9443.87 + - 114.84	7.81E-16 + - 5.91E-18	0.53 + - 0.06
4096	100	5	10	50	16	3	3.55 + - 0.37	0.0005 + - 6.86E-05	0.25 + - 0.05
4096	500	5	10	50	16	3	11.15 + - 0.16	3.86E-05 + - 1.16E-05	0.04 + - 0.01
4096	1000	5	10	50	16	3	21.00 + - 0.16	2.48E-08 + - 1.46E-08	0.01 + - 0.01
4096	5000	5	10	50	16	3	108.84 + - 0.45	8.79E-16 + - 9.86E-18	0.2 + - 0.03
4096	10000	5	10	50	16	3	241.97 + - 1.08	8.37E-16 + - 7.79E-18	0.18 + - 0.03
4096	50000	5	10	50	16	3	2760.29 + - 11.64	7.78E-16 + - 1.16E-17	0.51 + - 0.04
4096	100000	5	10	50	16	3	9482.29 + - 30.01	7.81E-16 + - 7.37E-18	0.53 + - 0.05
4096	100	5	15	50	16	3	3.16 + - 0.01	0.0005 + - 6.51E-05	0.21 + - 0.03
4096	500	5	15	50	16	3	11.3 + - 0.11	2.92E-05 + - 5.31E-06	0.03 + - 0.01
4096	1000	5	15	50	16	3	21.43 + - 0.06	8.43E-08 + - 1.08E-07	0.02 + - 0.02
4096	5000	5	15	50	16	3	140.39 + - 13.66	8.93E-16 + - 1.04E-17	0.18 + - 0.03

Map and Data Set Information				PSOM					
#Nodes	#Inputs	#Dim	Subsampled (%)	Iterat each Layer	Nodes in 1st Layer	Nr. Layers	Time	QE	TE
4096	10000	5	15	50	16	3	255.03 + - 3.4	8.34E-16 + - 6.48E-18	0.17 + - 0.02
4096	50000	5	15	50	16	3	2789.69 + - 13.39	7.77E-16 + - 3.37E-18	0.49 + - 0.03
4096	100000	5	15	50	16	3	9516.41 + - 89.38	7.63E-16 + - 1.18E-17	0.5 + - 0.08
4096	100	5	20	50	16	3	3.21 + - 0.01	0.0005 + - 6.18E-05	0.18 + - 0.04
4096	500	5	20	50	16	3	11.69 + - 0.18	2.49E-05 + - 4.52E-06	0.05 + - 0.02
4096	1000	5	20	50	16	3	21.95 + - 0.2	3.58E-08 + - 2.45E-08	0.02 + - 0.01
4096	5000	5	20	50	16	3	113.86 + - 0.68	8.84E-16 + - 1.37E-17	0.15 + - 0.01
4096	10000	5	20	50	16	3	252.25 + - 1.02	8.29E-16 + - 9.75E-18	0.19 + - 0.02
4096	50000	5	20	50	16	3	2808.49 + - 12.1	7.79E-16 + - 8.09E-18	0.46 + - 0.06
4096	100000	5	20	50	16	3	9475.16 + - 87.10	7.73E-16 + - 1.15E-17	0.54 + - 0.01
4096	100	5	25	50	16	3	3.16 + - 0.01	0.0005 + - 5.27E-05	0.21 + - 0.06
4096	500	5	25	50	16	3	11.48 + - 0.17	2.97E-05 + - 6.6E-06	0.06 + - 0.03
4096	1000	5	25	50	16	3	21.52 + - 0.07	8.26E-08 + - 9.09E-08	0.02 + - 0.02
4096	5000	5	25	50	16	3	134.81 + - 8.31	8.84E-16 + - 1.07E-17	0.2 + - 0.02
4096	10000	5	25	50	16	3	248.82 + - 1.07	8.35E-16 + - 6.42E-18	0.17 + - 0.01
4096	50000	5	25	50	16	3	2820.86 + - 8.38	7.78E-16 + - 5.89E-18	0.51 + - 0.02
4096	100000	5	25	50	16	3	9531.87 + - 116.77	7.70E-16 + - 7.28E-18	0.48 + - 0.03

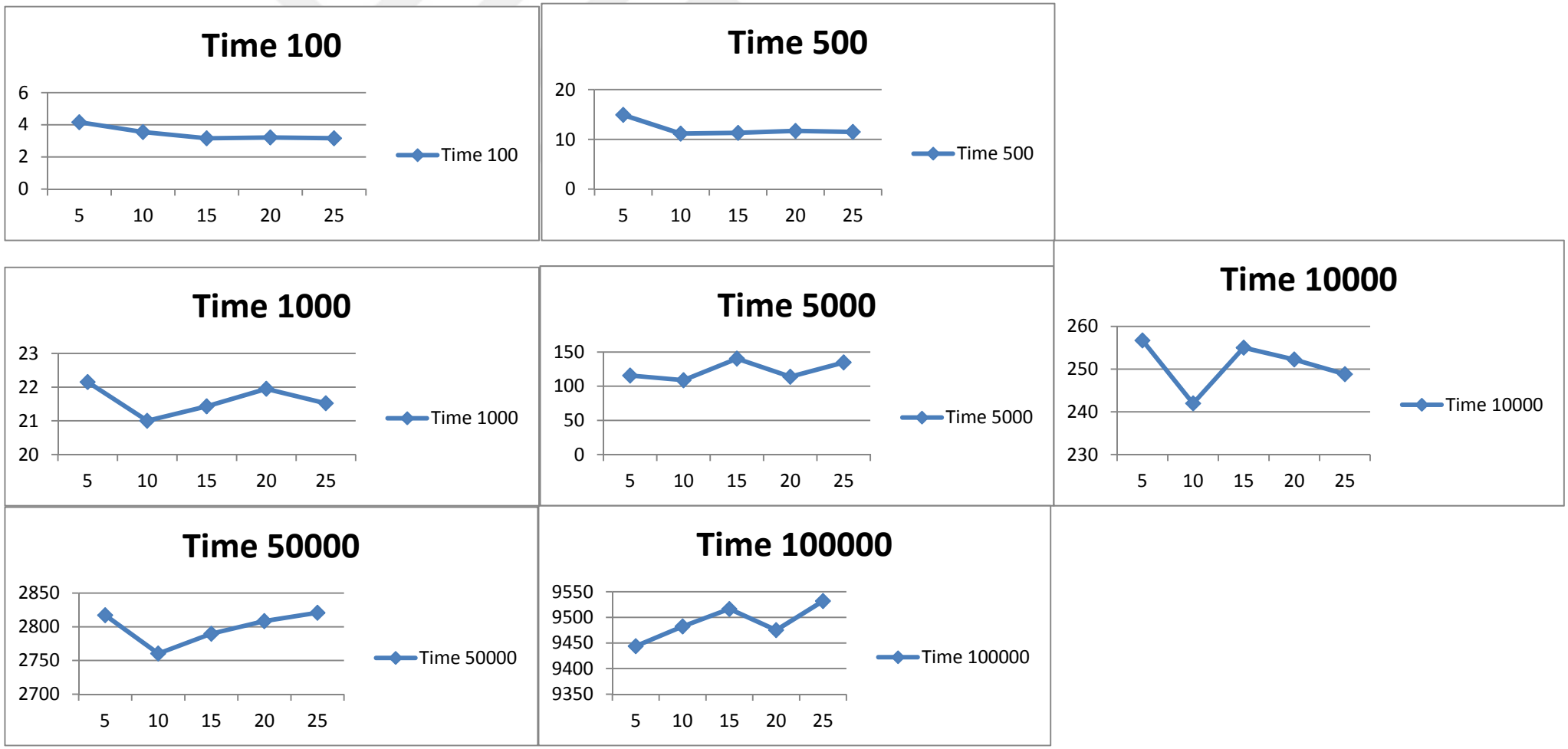
Appendix 3. Test results and SDOM values for PB-SOM Algorithm

Map and Data Set Information				PB - PSOM					
#Nodes	#Inputs	#Dim	Subsampled (%)	Iterations per Layer	Nodes in 1st Layer	Nr. Layers	Time (seconds)	QE	TE
4096	100	5	5	50	16	3	2.82 + - 0.06	5.27E-19 + - 9.4E-19	0.61 + - 0.054
4096	500	5	5	50	16	3	6.64 + - 0.03	2.27E-17 + - 1.74E-18	0.85 + - 0.03
4096	1000	5	5	50	16	3	11.55 + - 0.06	6.13E-17 + - 3.37E-18	0.86 + - 0.03
4096	5000	5	5	50	16	3	51.35 + - 0.12	3.42E-16 + - 9.25E-18	0.92 + - 0.02
4096	10000	5	5	50	16	3	104.41 + - 0.3	6.73E-16 + - 1.19E-17	0.89 + - 0.02
4096	50000	5	5	50	16	3	643.91 + - 5.57	3.49E-15 + - 1.43E-16	0.91 + - 0.006
4096	100000	5	5	50	16	3	1725.35 + - 19.08	1.13E-05 + - 2.43E-06	0.88 + - 0.02
4096	100	5	10	50	16	3	2.89 + - 0.05	2.3E-19 + - 3.32E-19	0.68 + - 0.03
4096	500	5	10	50	16	3	6.88 + - 0.14	2.39E-17 + - 1.18E-18	0.82 + - 0.02
4096	1000	5	10	50	16	3	11.87 + - 0.11	6.13E-17 + - 1.34E-18	0.89 + - 0.02
4096	5000	5	10	50	16	3	53.13 + - 0.42	3.38E-16 + - 6.24E-18	0.92 + - 0.01
4096	10000	5	10	50	16	3	107.54 + - 0.86	6.8E-16 + - 1.72E-17	0.90 + - 0.02
4096	50000	5	10	50	16	3	654.69 + - 3.96	2.02E-06 + - 3.8E-06	0.91 + - 0.009
4096	100000	5	10	50	16	3	1748.22 + - 16.09	5.39E-06 + - 3.92E-06	0.88 + - 0.01
4096	100	5	15	50	16	3	2.96 + - 0.09	3.33E-19 + - 3.67E-19	0.642 + - 0.0306724632202893
4096	500	5	15	50	16	3	6.92 + - 0.2	2.23E-17 + - 2.25E-18	0.8408 + - 0.0157419185615985
4096	1000	5	15	50	16	3	11.97 + - 0.12	6.47E-17 + - 2.86E-18	0.8834 + - 0.0166430766386507
4096	5000	5	15	50	16	3	52.44+ - 0.13	3.4E-16 + - 8.42E-18	0.92276 + - 0.03

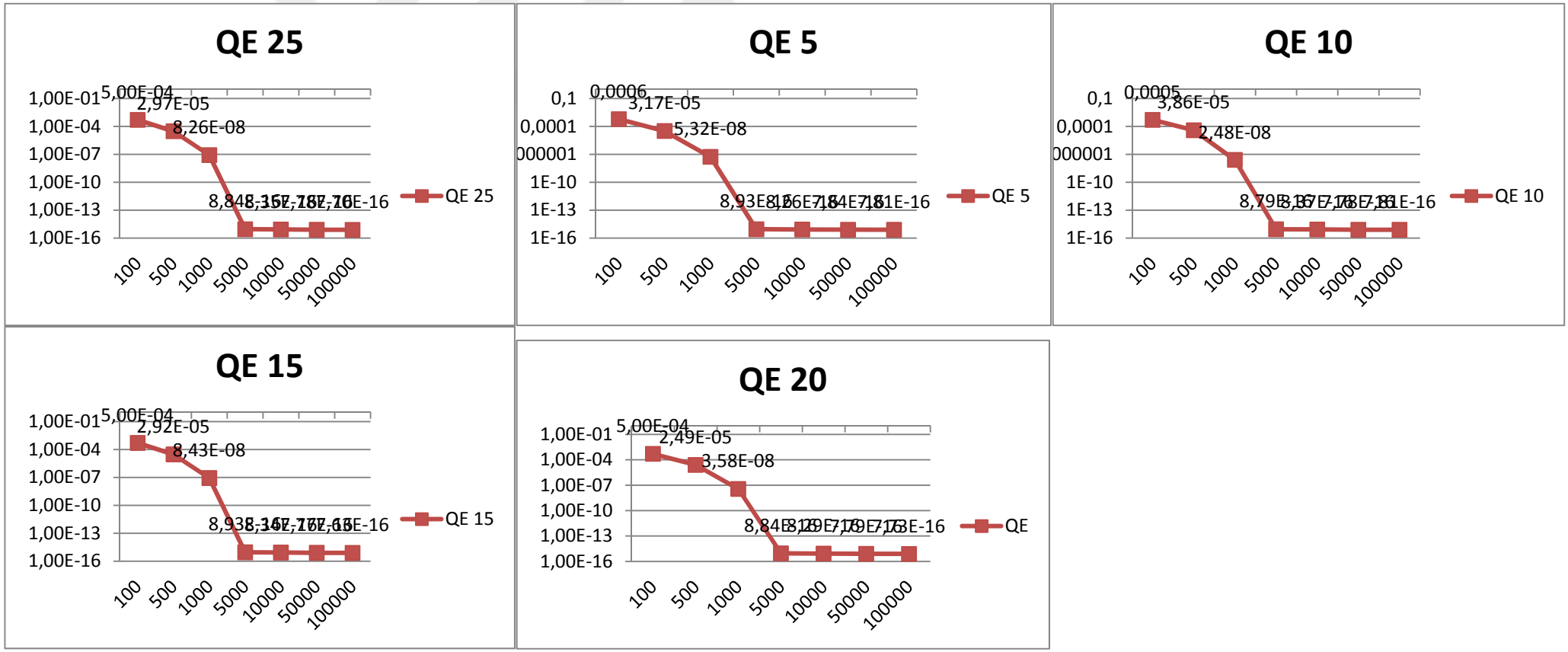
Map and Data Set Information				PB - PSOM					
#Nodes	#Inputs	#Dim	Subsampled (%)	Iterations per Layer	Nodes in 1st Layer	Nr. Layers	Time (seconds)	QE	TE
4096	10000	5	15	50	16	3	106.19 + - 0.55	6.83E-16 + - 1.43E-17	0.89338 + - 0.03
4096	50000	5	15	50	16	3	657.39 + - 2.41	3.58E-15 + - 1.96E-16	0.873188 + - 0.02
4096	100000	5	15	50	16	3	1751.12 + - 7.07	6.35E-06 + - 3.88E-06	0.846296 + - 0.02
4096	100	5	20	50	16	3	2.95 + - 0.06	0.00E+00	0.67 + - 0.06
4096	500	5	20	50	16	3	6.77 + - 0.15	2.31E-17 + - 2.19E-18	0.82 + - 0.01
4096	1000	5	20	50	16	3	11.67 + - 0.04	6.41E-17 + - 3.8E-18	0.87 + - 0.01
4096	5000	5	20	50	16	3	51.98 + - 0.14	3.35E-16 + - 9.6E-18	0.92 + - 0.02
4096	10000	5	20	50	16	3	105.47 + - 0.81	6.79E-16 + - 1.67E-17	0.93 + - 0.02
4096	50000	5	20	50	16	3	660.41 + - 0.49	3.54E-15 + - 8.93E-17	0.89 + - 0.03
4096	100000	5	20	50	16	3	1786.574 + - 9.71	5.93E-07 + - 8.39E-07	0.88 + - 0.04
4096	100	5	25	50	16	3	2.89+ - 0.02	0.00E+00	0.71 + - 0.07
4096	500	5	25	50	16	3	6.97 + - 0.17	2.23E-17 + - 2.23E-18	0.84 + - 0.03
4096	1000	5	25	50	16	3	11.81 + - 0.04	5.88E-17 + - 1.38E-18	0.89 + - 0.02
4096	5000	5	25	50	16	3	53.04 + - 0.294	3.264E-16 + - 8.53E-18	0.92 + - 0.01
4096	10000	5	25	50	16	3	107.39 + - 0.31	6.86E-16 + - 1.17E-17	0.93 + - 0.02
4096	50000	5	25	50	16	3	682.91 + - 1.01	3.48E-15 + - 7.61E-17	0.90 + - 0.02
4096	100000	5	25	50	16	3	1876.51 + - 8.59	6.11E-06 + - 4.02E-06	0.88 + - 0.02



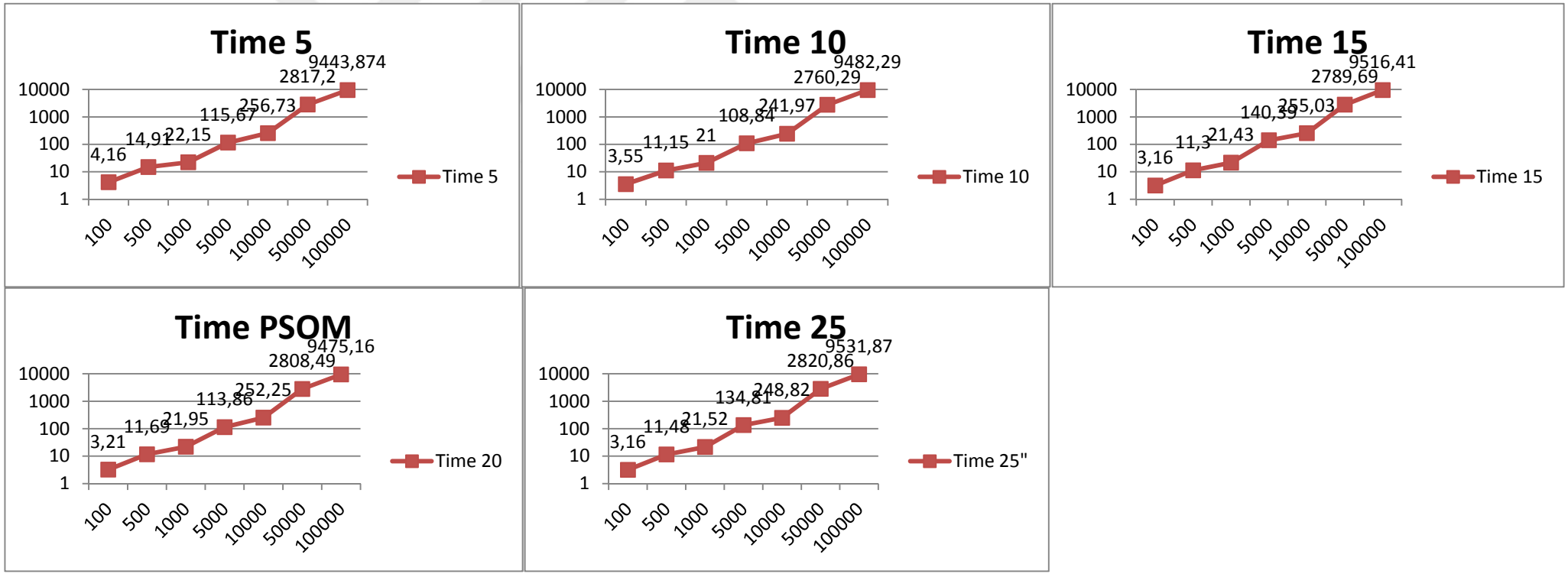
Appendix 4. Quantization Error of PSOM according to subsample size in the first map, for every input size



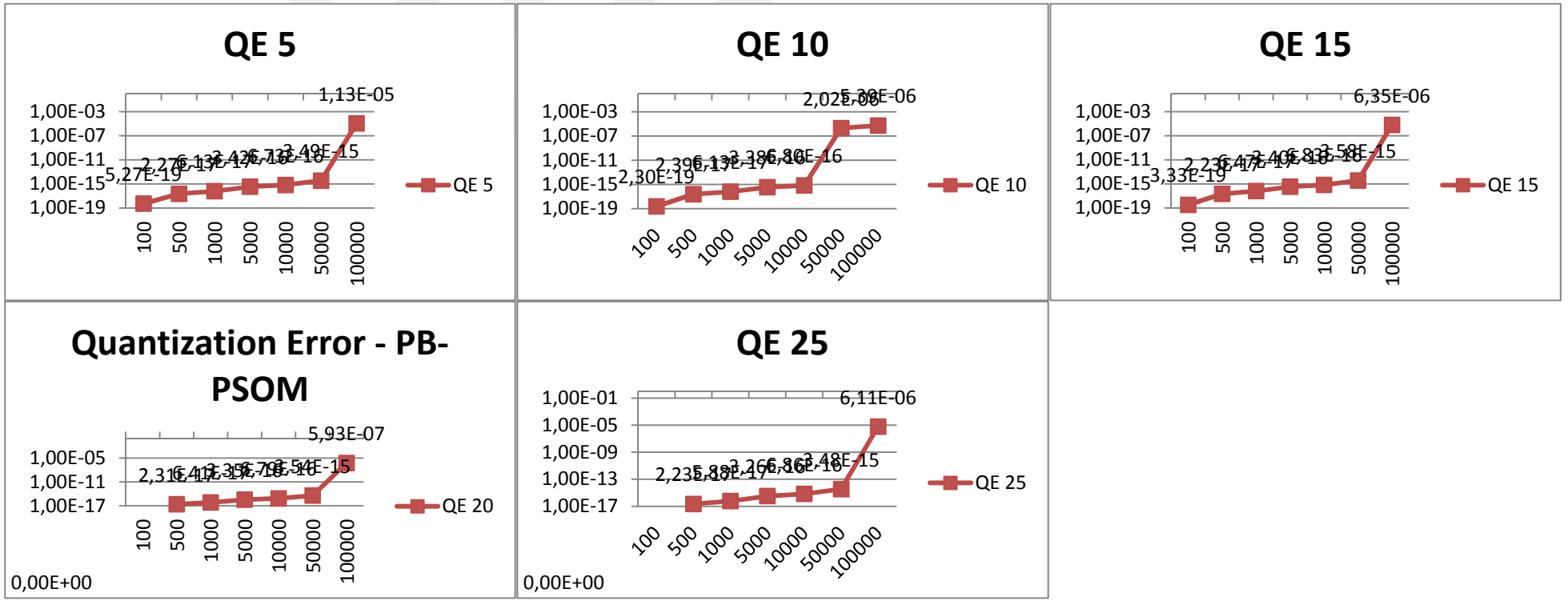
Appendix 5. Execution Time of PSOM according to subsample size in the first map, for every input size.



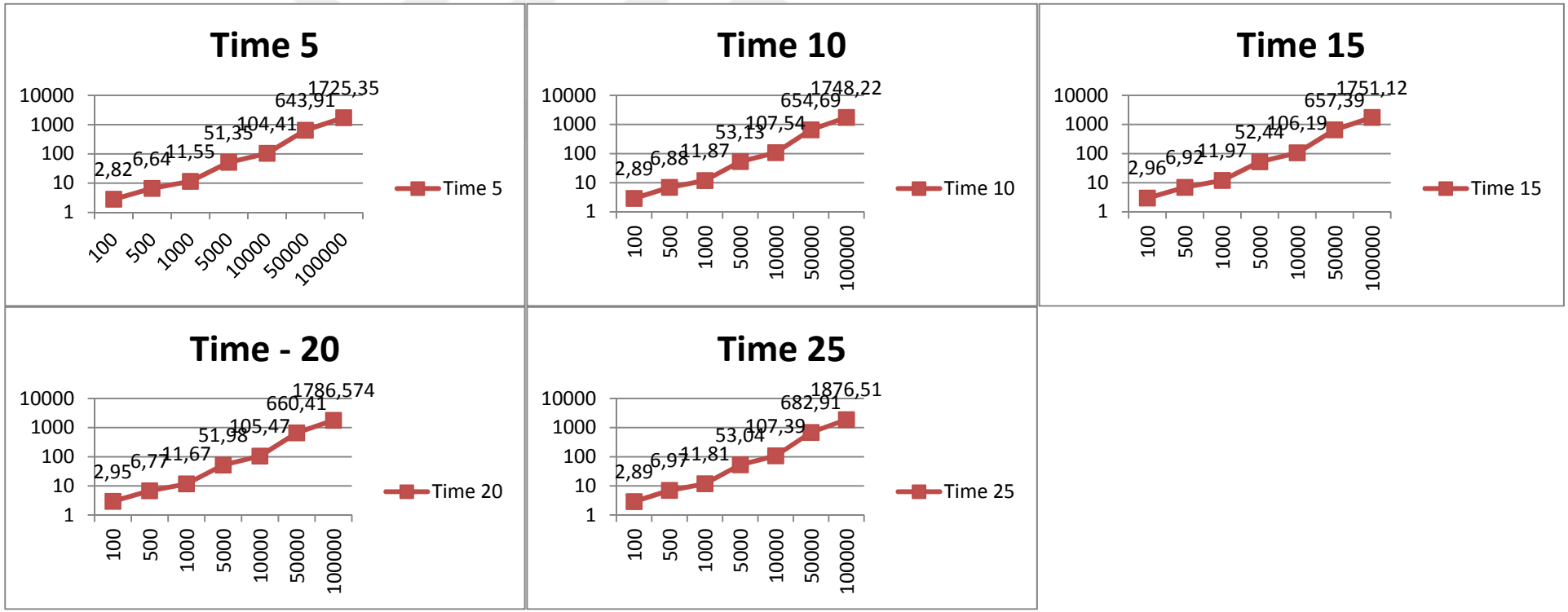
Appendix 6. Quantization Error of PSOM according to size of input data, for every subsample size.



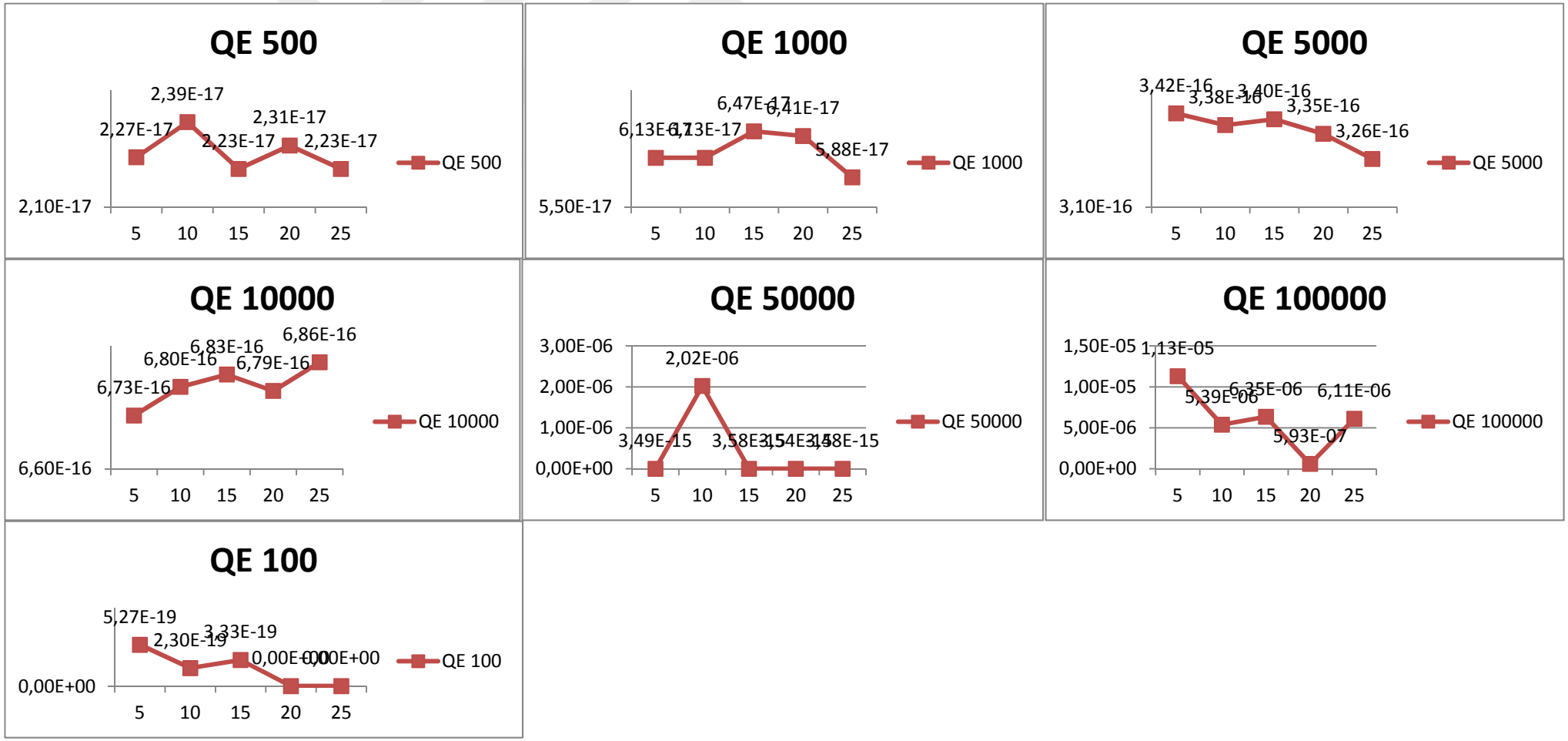
Appendix 7. Execution Time of PSOM according to size of input data, for every subsample size.



Appendix 8. Quantization Error of PB-PSOM according to size of input data, for every subsample size.



Appendix 9. Execution Time of PB-PSOM according to size of input data, for every subsample size.



Appendix 10. Quantization Error of PB-PSOM according to subsample size in the first map, for every input size.