Self Organizing Maps Whose Topologies Can Be Learned With Adaptive Binary Search Trees Using Conditional Rotations

César A. Astudillo * † B. John Oommen[‡] §

Abstract

Numerous variants of Self-Organizing Maps (SOMs) have been proposed in the literature, including those which also possess an underlying structure, and in some cases, this structure itself can be defined by the user Although the concepts of growing the SOM and updating it have been studied, the whole issue of using a self-organizing Adaptive Data Structure (ADS) to further enhance the properties of the underlying SOM, has been unexplored. In an earlier work, we impose an *arbitrary*, user-defined, tree-like topology onto the codebooks, which consequently enforced a neighborhood phenomenon and the so-called tree-based Bubble of Activity (BoA). In this paper, we consider how the underlying tree itself can be rendered dynamic and adaptively transformed. To do this, we present methods by which a SOM with an underlying Binary Search Tree (BST) structure can be adaptively re-structured using Conditional Rotations (CONROT). These rotations on the nodes of the tree are local, can be done in constant time, and performed so as to decrease the Weighted Path Length (WPL) of the entire tree. In doing this, we introduce the pioneering concept referred to as Neural Promotion, where neurons gain prominence in the Neural Network (NN) as their significance increases. We are not aware of any research which deals with the issue of Neural Promotion. The advantages of such a scheme is that the user need not be aware of any of the topological peculiarities of the stochastic data distribution. Rather, the algorithm, referred to as the TTOSOM with Conditional Rotations (TTOCONROT), converges in such a manner that the neurons are ultimately placed in the input space so as to represent its stochastic distribution, and additionally, the neighborhood properties of the neurons suit the best BST that represents the data. These properties have been confirmed by our experimental results on a variety of data sets. We submit that all of these concepts are both novel and of a pioneering sort.

Keywords: Adaptive Data Structures, Binary Search Trees, Self-Organizing Maps

^{*}Universidad de Talca, Merced 437 Curicó, Chile. castudillo@utalca.cl

[†]This author is Assistant Professor at the Department of Computer Science, with the Universidad de Talca. This work is partially supported by the FONDECYT grant 11121350, Chile. A very preliminary version of this paper was presented at AI'09, the 2009 Australasian Joint Conference on Artificial Intelligence, Melbourne, Australia, in December 2009. That paper won the award of being the *Best Paper of the Conference*. We are also very grateful for the comments made by the Associate Editor and the anonymous Referees. Their input helped in improving the quality of the final version of this paper. Thank you very much! [‡]School of Computer Science, Carleton University, Ottawa, Canada : K1S 5B6. commen@scs.carleton.ca

[§]Chancellor's Professor; Fellow : IEEE and Fellow : IAPR. This author is also an Adjunct Professor with the University of Agder in Grimstad, Norway. The work of this author was partially supported by NSERC, the Natural Sciences and Engineering Research Council of Canada.

1 Introduction

This paper is a pioneering attempt to merge the areas of Self-Organizing Maps (SOMs) with the theory of Adaptive Data Structures (ADSs). Put in a nutshell, we can describe the goal of this paper as follows: Consider a SOM with n neurons. Rather than having the neurons merely possess information about the feature space, we also attempt to *link* them together by means of an underlying Data Structure (DS). This DS could be a singly-linked list, a doubly-linked list or a Binary Search Tree (BST), etc. The intention is that the neurons are governed by the laws of the SOM and the underlying DS. Observe now that the concepts of "neighborhood" and Bubble of Activity (BoA) are not based on the nearness of the neurons in the feature space, but rather on their proximity in the underlying DS. Having accepted the above-mentioned premise, we intent to take this entire concept to a higher level of abstraction and propose to modify this DS *itself* adaptively using operations specific to it. As far as we know, the combination of these concepts has been unreported in the literature.

Before we proceed, to place our results in the right perspective, it is probably wise to see how the concept of neighborhood has been defined in the SOM literature.

Kohonen, in his book [36], mentions that it is possible to distinguish between two basic types of neighborhood functions. The first family involves a kernel function (which is usually of a Gaussian nature). The second, is the so-called neighborhood set, also known as the Bubble of Activity (BoA). This paper focuses on the second type of neighborhood function.

Even though the traditional SOM is dependent on the neural distance to estimate the subset of neurons to be incorporated into the BoA, this is not always the case for the SOM-variants included in the literature. Indeed, the different strategies described in the state-of-the-art utilize families of schemes to define the BoA. We mainly identify three sub-classes.

The first type of BoA uses the concept of the neural distance as in the case of the traditional SOM. Once the Best Matching Unit (BMU) is identified, the neural distance is calculated by traversing the underlying structure that holds the neurons. An important property of the neural distance between two neurons is that it is proportional to the number of connections separating them. Examples of strategies that use the neural distance to determine the BoA are the Growing Cell Structures (GCS) [24], the Growing Grid (GG) [25], the Incremental Grid Growing (IGG) [13], the Growing SOM (GSOM) [3], the Tree-Structured SOM (TSSOM) [37], the Hierarchical Feature Map (HFM) [43], the Growing Hierarchical SOM (GHSOM) [50], the Self-Organizing Tree Algorithm (SOTA) [22], the Evolving Tree (ET) [46], the Tree-based Topology Oriented SOM (TTOSOM) [8], among others.

A second subset of strategies employ a scheme for determining the BoA that does not depend on the inter-neural connections. Instead, such strategies utilize the distance in the feature space. In these cases, it is possible to distinguish between two types of Neural Networks (NNs). The simplest situation occurs when the BoA only considers the BMU, i.e., it constitutes an instance of hard Competitive Learning (CL), as in the case of the Tree-Structured VQ (TSVQ) [37] and the Self-Organizing Tree Map (SOTM) [27].

A more sophisticated and computationally expensive scheme involves ranking the neurons as per their respective distances to the stimulus. In this scenario, the BoA is determined by selecting a subset of the closest neurons. An example of a SOM variant that uses such a ranking is the Neural Gas (NG) [40].

According to the authors of [46], the SOM-based variants included in the literature attempt to tackle two main goals: They either try to design a more flexible topology, which is usually useful to analyze large datasets, or to reduce the the most time-consuming task required by the SOM, namely, the search for the BMU when the input set has a complex nature. In this paper we focus on the former of the two mentioned goals. In other words, our goal is to enhance the capabilities of the original SOM algorithm so as to represent the underlying data distribution and its structure in a more accurate manner. We also intend to do so by constraining the neurons so that they are related to each other, *not just based on their neural indices and stochastic distribution*, but also based on a BST relationship.

Furthermore, as a long term ambition, we also anticipate methods which can accelerate the task of locating the nearest neuron during the CL phase. This work will present the details of the design and implementation of how an adaptive process applied to the BST, can be integrated into the SOM.

Regardless of the fact that numerous variants of the SOM has been devised, few of them possess the ability of modifying the underlying topology [13, 21, 22, 26, 27, 42, 46, 52]. Moreover, only a small subset use a tree as their underlying DS [8, 21, 22, 27, 46, 52]. These strategies attempt to dynamically modify the nodes of the SOM, for example, by adding nodes, which can be a single neuron or a layer of a SOM-grid. However, our hypothesis is that it is also possible to attain to a better understanding of the unknown data distribution by performing *structural* tree-based modifications on the tree, which although they preserve the general topology, attempt to modify the overall configuration, i.e., by altering the way by which nodes are *interconnected*, and yet continue as a BST. We accomplish this by dynamically adapting the edges that connect the neurons, by rotating¹ the nodes within the BST that holds the whole structure of neurons. As we will explain later, this is further achieved by local modifications to the overall structure in a constant number of steps. Thus, we attempt to use rotations, tree-based neighbors *and* the feature space to improve the quality of the SOM.

1.1 Motivations

Acquiring information about a set of stimuli in an unsupervised manner, usually demands the deduction of its structure. In general, the *topology* employed by any Artificial Neural Network (ANN) possessing this ability has an important impact on the manner by which it will "absorb" and display the properties of the input set. Consider for example, the following: A user may want to devise an algorithm that is capable of learning a triangle-shaped distribution as the one depicted in Figure 1. The SOM tries to achieve this by defining an underlying grid-based topology and to fit the grid within the overall shape, as shown in Figure 1a (duplicated from [36]). However, from our perspective, a grid-like topology does not naturally fit a triangular-shaped distribution, and thus, one experiences a deformation of the original lattice during the modeling phase. As opposed to this, Figure 1b, shows the result of applying one of the techniques developed by us, namely the TTOSOM [8]. As the reader can observe from Figure 1b, a 3-ary tree seems to be a far more superior choice for representing the particular shape in question.

¹The operation of rotation is the one associated with BSTs, as will be presently explained.



Figure 1: How a triangle-shaped distribution is learned through unsupervised learning.

On closer inspection, Figure 1b depicts how the complete tree fills in the triangle formed by the set of stimuli, and further, seems to do it *uniformly*. The final position of the nodes of the tree suggests that the underlying structure of the data distribution corresponds to the triangle. Additionally, the root of the tree is placed roughly in the center of mass of the triangle. It is also interesting to note that each of the three main branches of the tree, cover the areas directed towards a vertex of the triangle respectively, and their sub-branches fill in the surrounding space around them in a recursive manner, which we identify as being a holograph-like behavior.

Of course, the triangle of Figure 1b serves only as a very simple *prima facie* example to demonstrate to the reader, in an informal manner, how both techniques will try to learn the set of stimuli. Indeed, in real-world problems, these techniques can be employed to extract the properties of high-dimensional samples.

One can argue that imposing an initial topological configuration is not in accordance with the founding principles of unsupervised learning, the phenomenon that is supposed to occur without "supervision" within the human brain. As an initial response we argue that this "supervision" is required to enhance the *training* phase, while the information we provide relates to the *initialization* phase. Indeed, this is in line with the well-accepted principle [23], that very little can be automatically learned about a data distribution if no assumptions are made!

As the next step of motivating this research endeavor, we venture into a world where the neural topology *and* structure are themselves learned during the training process. This is achieved by the method that we propose in this paper, namely the TTOSOM with Conditional Rotations (TTOCONROT), which, in essence, dynamically extends the properties of the above-mentioned TTOSOM. Again, to accomplish this we need key concepts that are completely new to the field of SOMs, namely those related to tree-based Adaptive Data Structure (ADS). Indeed, as demonstrated by our experiments, the results that we have already obtained have been applauded by the research community², and these, to the best of our knowledge, have remained unreported in the literature.

Another reason why we are interested in such an inter-area integration, deals with the issue for devising efficient methods that add neurons to the tree. Even though the schemes that we are currently proposing

 $^{^{2}}$ As mentioned earlier, a paper which reported the preliminary results of this study, won the *Best Paper Award* in a well-known international AI conference [7].

in this paper focus on tree adaptation by means of rotations, we envision another type of dynamism, i.e., one which involves the expansion of the tree structure through the insertion of newly created nodes. The state-of-the-art considers different strategies that expand trees by inserting nodes (which can be a single neuron or a SOM-layer) that essentially are based on a Quantization Error (QE) measure. In some of these strategies, the error measure is based on the "hits", i.e., the number of times a neuron has been selected as the BMU, c.f., [13, 24, 37, 46]. The strategy that we have chosen for adapting the tree, namely using Conditional Rotations (CONROT), already utilizes this BMU counter, and, distinct to the previous strategies that attempt to search for a node to be expanded (which in the case of tree-based SOMs is usually at the level of the leaves [37, 46]), we foresee and advocate a different approach. Our TTOCONROT method asymptotically positions frequently accessed nodes close to the root, and so, according to this property, it is the root node which should be split. Observe that if we follow such a philosophy, one would not have to search for a node with a higher QE measure. Rather, the CONROT, will be hopefully, able to migrate the candidates closer to the root. Of course, this works with assumption that a larger number of hits indicates that the degree of granularity of a particular neuron justifies refinement. The concept of using the root of the tree for growing a tree-based SOM is, in and of itself, pioneering, as far as we know.

1.2 Contributions of the Paper

The contributions of the paper can be summarized as follows:

- 1. We present an integration of the fields of SOMs and ADS. This, we respectfully, submit as pioneering.
- 2. The neurons of the SOM are linked together using an underlying tree-based DS, and they are governed by the laws of the TTOSOM tree-based paradigm, and simultaneously the restructuring adaptation provided by CONROT.
- 3. The definition of distance between the neurons is based on the tree structure, and not in the feature space. This is valid also for the BoA, rendering the migrations distinct from the state-of-the-art.
- 4. The adaptive nature of the TTOCONROT is unique because adaptation is perceived in two forms: The migration of the codebook vectors in the feature space is a consequence of the SOM update rule, and the rearrangement of the neurons *within* the tree as a result of the rotations.

1.3 Organization of the Paper

The rest of the paper is organized as follows. The next section surveys the relevant literature³, which involves both the field of SOMs including their tree-based instantiations, and the respective field of BSTs with conditional rotations. After that, in Section 2, we provide an in-depth explanation of the TTOCONROT philosophy, which is our primary contribution. The subsequent section shows the capabilities of the approach through a series of experiments, and finally, Section 5 concludes the paper.

 $^{^{3}}$ For the sake of space the literature review has been considerably condensed. However, given that there is no survey paper on the area of tree-based SOMs reported in the literature, we are currently preparing a paper that summarizes the field.

2 Literature Review

2.1 The SOM

One of the most important families of ANNs used to tackle clustering problems is the well known SOM [36]. Typically, the SOM is trained using (un)supervised learning, so as to produce a neural representation in a space whose dimension is usually smaller than that in which the training samples lie. Further, the neurons attempt to preserve the topological properties of the input space.

The SOM concentrates all the information contained in a set of n input samples belonging to the ddimensional space, say $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, utilizing a much smaller set of neurons, $\mathcal{C} = \{\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_m\}$, each of which is represented as a vector. Each of the *m* neurons contains a weight vector $\mathbf{w} = [w_1, w_2, \dots, w_d]^t \in$ \mathbb{R}^d associated with it. These vectors are synonymously called "weights", "prototypes" or "codebook" vectors. The vector \mathbf{w}_i may be perceived as the *position* of neuron \mathbf{c}_i in the feature space. During the training phase, the values of these weights are adjusted simultaneously so as to represent the data distribution and its structure. In each training step a stimulus (a representative input sample from the data distribution) \mathbf{x} is presented to the network, and the neurons compete between themselves so as to identify which is the "winner", also known as the Best Matching Unit (BMU). After identifying the BMU, a subset of the neurons "close" to it are considered to be within the so-called Bubble of Activity (BoA), which further depends on a parameter specified to the algorithm, namely, the so-called radius. Thereafter, this scheme performs a migration of the codebooks within that BoA so as to position them closer to the sample being examined. The migration factor by which this update is effected, depends on a parameter known as the learning rate, which is typically expected to be large initially, and which decreases as the algorithm proceeds, and which ultimately results in no migration at all. Algorithm 1 describes the details of the SOM philosophy. In Algorithm 1, the parameters are scheduled by defining a sequence $\mathcal{S} = \langle S_1, S_2, \ldots, S_s \rangle$, where each S_i corresponds to a tuple (η_i, r_i, t_i) that specifies the learning rate, η_i , and the radius, r_i , for a fixed number of training steps, t_i . The way in which the parameters decay is not specified in the original algorithm, and some alternatives are, e.g., that the parameters remain fixed, decrease linearly, exponentially, etc.

Algorithm 1 $SOM(\mathcal{X}, \mathcal{S})$

Input:

i) \mathcal{X} , the input sample set.

ii) S, the schedule for the parameters.

Method:

1: Initialize the weights $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m$, e.g., by randomly selecting elements from \mathcal{X} .

2: repeat

- 3: Obtain a sample \mathbf{x} from \mathcal{X} .
- 4: Find the Winner neuron, i.e., the one which is most similar to \mathbf{x} .
- 5: Determine a subset of neurons close to the winner.
- 6: Migrate the closest neuron and its neighbors towards \mathbf{x} .
- 7: Modify the learning factor and radius as per the pre-defined schedule.
- 8: until no noticeable changes are observed.

End Algorithm

Although the SOM has demonstrated an ability to solve problems over a wide spectrum, it possesses some

fundamental drawbacks. One of these drawbacks is that the user must specify the lattice *a priori*, which has the effect that he must run the ANN a number of times to obtain a suitable configuration. Other handicaps involve the size of the maps, where a lesser number of neurons often represent the data inaccurately.

The state-of-the-art approaches attempt to render the topology more flexible, so as to represent complicated data distributions in a better way and/or to make the process faster by, for instance, speeding up the task of determining the BMU.

There are a vast number of domain fields where the SOM has demonstrated to be useful; a compendium with all the articles that take advantage of the properties of the SOM is surveyed in [32, 45]. These survey papers classify the publications related to the SOM according to their year of release. The report [32] includes the bibliography published between the year 1981 and 1998, while the report [45] includes the analogous papers published between 1998 and 2001. Further, additional recent references including the related work up to the year 2005 have been collected in a technical report [48]. The more recent literature reports a host of application domains, including Medical Image Processing [2], Human Eye Detection [33], Handwriting Recognition [39], Image Segmentation [56], Information Retrieval [20], Object Tracking [30], etc.

2.2 Tree-Based SOMs

Although an important number of variants of the original SOM have been presented through the years, we focus our attention on a specific family of enhancements in which the neurons are inter-connected using a tree topology.

The Tree-Structured VQ (TSVQ) algorithm [37] is a tree-based SOM variant, whose topology is defined a priori and which is static. The training first takes place at highest levels of the tree. The TSVQ incorporates the concept of a "frozen" node, which implies that after a node is trained for a certain amount of time, it becomes static. The algorithm then allows subsequent units, i.e., the direct children, to be trained. The strategy utilizes a heuristic search algorithm for rapidly identifying a BMU. It starts from the root and recursively traverses the path towards the leaves. If the unit currently being analyzed is frozen, the algorithm identifies the child which is closest to the stimulus, and performs a recursive call. The algorithm terminates when the node currently being analyzed is not a frozen node (i.e., it is currently being trained), and is returned as the BMU.

Koikkalainen and Oja, in the same paper [37] refine the idea of the TSVQ by defining the TSSOM, which inherits all the properties of the TSVQ, but redefines the search procedure and BoA. In the case of the TSSOM, SOM layers of different dimensions are arranged in a pyramidal shape (which can be perceived as a SOM with different degrees of granularity). It differs from the TSVQ, in the sense that, once the BMU is found, the direct proximity is examined to check for the BMU. On the other hand, the BoA differs in that, instead of considering only the BMU, its direct neighbors (in the pyramid) will also be considered.

The Self-Organizing Tree Algorithm (SOTA) [22] is a dynamically growing tree-based SOM which, according to their authors, take some analogies from the Growing Cell Structures (GCS) [24]. The SOTA utilizes a binary tree as the underlying structure, and similarly to other strategies (e.g., the TSSOM [37] and the Evolving Tree (ET) [46] explained below), it considers the migration of the neurons *only* if they correspond to leaf nodes within the tree structure. Its BoA depends on the neural tree and is defined for two cases. The most general case occurs when the parent of the BMU is not the root, i.e., a situation in which the BoA is composed by the BMU, its sibling and its parent node. Otherwise, the BoA constitutes the BMU only. The SOTA triggers a growing mechanism that utilizes a QE to determine the node to be split into two new descendants.

In [21] the authors presented a tree-based SOM called the Growing Hierarchical SOM (GHSOM), in which each node corresponds to an independent SOM. The expansion of the structure is dual: The first type of adaptation is conceived by inserting new rows (or columns) to the SOM grid that is currently being trained, while the second type is implemented by adding layers to the hierarchical structure. Both types of dynamism depend on the verification of QE measures.

The SOTM [27] is a tree-based SOM which is also inspired by the Adaptive Resonance Theory (ART) [15]. In the SOTM, when the input is within a threshold distance from the BMU, the latter is migrated. Otherwise, a new neuron is added to the tree. Thus, in the SOTM, the subset of neurons to be migrated depends only on the distance in the *feature* space, and not in the neural distance, as most of the tree-based SOM families.

In [46], the authors have proposed a tree-structured NN called the Evolving Tree (ET), which takes advantage of a sub-optimal procedure (adapted from the one utilized by the TSVQ) to identify the BMU in $O(\log |V|)$ time, where V is the set of neurons. The ET adds neurons dynamically, and incorporates the concept of a "frozen" neuron (explained above), which is a non-leaf node that does not participate in the training process, and which is thus removed from the BoA. Similar to the TSVQ, the training phase terminates when all the nodes become frozen.

The Tree-based Topology Oriented SOM (TTOSOM) [8], which is central to this paper, is a tree-based SOM in which each node can possess an arbitrary number of children. Furthermore, it is assumed that the user has the ability to describe/create such a tree whose topological configuration is preserved through the training process. The TTOSOM uses a particular BoA that includes nodes (leaf and non-leaf ones) that are within a certain neural distance (the so-called "radius"). An interesting property displayed by this strategy is its ability to reproduce the results obtained by Kohonen, when the nodes of the SOM are arranged linearly, i.e., in a list. In this case, the TTOSOM is able to adapt this 1-dimensional grid to a 2-dimensional (or multi-dimensional) object in the same way as the SOM algorithm does [8]. This was a phenomenon that was not possessed by prior hierarchical SOM-based networks reported in the literature⁴. Additionally, if the original topology of the tree followed the overall shape of the data distribution, the results reported in [8] (and also depicted in the motivational section) showed that is also possible to obtain a symmetric topology for the codebook vectors. In a more recent work [9], the authors have enhanced the TTOSOM to perform classification in a semi-supervised fashion. The method presented in [9] first learns the data distribution in an unsupervised manner. Once labeled instances become available, the clusters are labeled using the evidence. According to the results presented in [9], the number of neurons required to accurately predict the category

 $^{^{4}}$ The SOM possesses the ability to learn the data distribution by utilizing a unidimensional topology [36], i.e., the neighbors are defined along a grid in each direction. Further, when this is the case, one can encounter that the unidimensional topology forms a so-called *Peano* curve [47]. The TTOSOM also possesses this interesting property, when the tree topology is linear. The details of how this is achieved is presented in detail in [8], including the explanation of how other tree-based techniques fail to achieve this task.

of novel data are only a small portion of the cardinality of the input set.

3 Merging ADS and TTOSOM

3.1 Adaptive Data Structures (ADSs) for BSTs

One of the primary goals of the area of ADS is to achieve an optimal arrangement of the elements, placed at the nodes of the structure, as the number of iterations increases. This reorganization can be perceived to be both automatic and adaptive, such that on convergence, the DS tends towards an optimal configuration with a minimum average access time. In most cases, the most probable element will be positioned at the root (head) of the tree (DS), while the rest of the tree is recursively positioned in the same manner. The solution to obtain the *optimal* BST is well known when the access probabilities of the nodes are known a *priori* [35]. However, our research concentrates on the case when these access probabilities are *not known a priori*. In this setting, one effective solution is due to Cheetham *et al.* and uses the concept of CONROT [16], which reorganizes the BST so as to asymptotically produce the optimal form. Additionally, unlike most of the algorithms that are otherwise reported in the literature, this move is not done on every data access operation – it is performed if and only if the overall Weighted Path Length (WPL) of the resulting BST decreases.

A BST may be used to store records whose keys are members of an ordered set $\mathcal{A} = \{A_1, A_2, \ldots, A_N\}$. The records are stored in such a way that a symmetric-order traversal of the tree will yield the records in an ascending order. If we are given \mathcal{A} and the set of access probabilities $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_N\}$, the problem of constructing efficient BSTs has been extensively studied. The optimal algorithm due to Knuth [35], uses dynamic programming and produces the optimal BST using $O(N^2)$ time and space. In this paper, we consider the scenario in which \mathcal{Q} , the access probability vector, is not known *a priori*. We seek a scheme which dynamically rearranges itself and asymptotically generates a tree which minimizes the access cost of the keys.

The primitive tree restructuring operation used in most BST schemes is the well known operation of Rotation [1]. We describe this operation as follows. Suppose that there exists a node i in a BST, and that it has a parent node j, a left child, i_L , and a right child, i_R . The function P(i) = j relates node i with its parent j (if it exists). Also, let B(i) = k relate node i with its sibling k, i.e., the node (if it exists) that shares the same parent as i. Consider the case when i is itself a left child (see Figure 2a). A rotation is performed on node i as follows: j now becomes the right child, i_R becomes the left child of node j, and all the other nodes remain in their same relative positions (see Figure 2b). The case when node i is a right child is treated in a symmetric manner. This operation has the effect of raising (or promoting) a specified node in the tree structure while preserving the lexicographic order of the elements (refer again to Figure2b).

A few memory-less tree reorganizing schemes⁵ which use this operation have been presented in the literature among which are the Move-to-Root and the simple Exchange rules [4]. In the Move-to-Root Heuristic, each time a record is accessed, rotations are performed on it in an upwards direction until it becomes the

⁵This review is necessary brief. A more detailed version is found in [18, 38].



formed. The contents of the nodes are their data values, which in this case are the characters $\{a, b, c, d, e\}$.

(b) The tree after a rotation is performed on node i.

Figure 2: The BST before and after a Rotation is performed.

root of the tree. On the other hand, the simple Exchange rule rotates the accessed element one level towards the root.

Sleator and Tarjan [54] introduced a technique, which also moves the accessed record up to the root of the tree using a restructuring operation called "Splaying", which actually is a multi-level generalization of the rotation. Their structure, called the Splay Tree, was shown to have an amortized time complexity of $O(\log N)$ for a complete set of tree operations which included insertion, deletion, access, split, and join.

The literature also records various schemes which adaptively restructure the tree with the aid of additional memory locations. Prominent among them is the Monotonic Tree (MT) [12] and Mehlhorn's D-Tree (DT) [41]. The MT is a dynamic version of a tree structuring method originally suggested by Knuth [35].

In spite of all their advantages, all of the schemes mentioned above have drawbacks, some of which are more serious than others. The memory-less schemes have one major disadvantage, which is that both the Move-to-Root and Splaying rules always move the accessed record up to the root of the tree. This means that if a nearly-optimal arrangement is reached, a single access of a seldomly-used record will disarrange the tree along the entire access path, as the element is moved upwards to the root.

As opposed to these schemes, the MT rule does not move the accessed element to the root every time. But, as reported in [16], in practice, it does not perform well. The weakness of the MT lies in the fact that it considers only the frequency counts for the records, which leads to the undesirable property that a single rotation may move a subtree with a relatively large probability weight downwards, thus increasing the cost of the tree.

This paper uses a particular heuristic, namely, the Conditional Rotations for a BST (CONROT-BST) [16], which has been shown to reorganize a BST so as to asymptotically arrive at an optimal form. In its optimized version, the scheme, referred to Algorithm CONROT-BST, requires the maintenance of a single memory location per record, which keeps track of the number of accesses to the subtree rooted at that record. The CONROT-BST algorithm specifies how an accessed element can be rotated towards the root of the tree so as to minimize the overall cost of the entire tree. Finally, unlike most of the algorithms that are currently

in the literature, this move is not done on every data access operation. It is performed if and only if the overall WPL of the resulting BST decreases. In essence Algorithm CONROT-BST attempts to minimize the WPL by incorporating the statistical information about the accesses to the various nodes *and subtrees* rooted at the corresponding nodes.

The basic condition for the rotation of a node is that the WPL of the entire tree must decrease as a result of a single rotation. This is achieved by a so-called *Conditional* Rotation. To define the concept of a Conditional Rotation, we define $\tau_i(n)$ as the total number of accesses to the subtree rooted at node *i*. One of the biggest advantages of the CONROT-BST heuristic is that it only requires the maintenance and processing of the values stored at a specific node and its direct neighbors, i.e., its parent and both children, if they exist.

Algorithm CONROT-BST, formally given in Algorithm 2, describes the process of the conditional rotations for a BST. The algorithm receives two parameters, the first of which corresponds to a pointer to the root of the tree, and the second which corresponds to the key to be searched, which is assumed to be present in the tree. When a node access is requested, the algorithm seeks for the node from the root down towards the leaves.

Algorithm 2 CONROT-BST (j,k_i)

```
Input:
i) j, A pointer to the root of a binary search tree T
ii) k_i, A search key, assumed to be in T
Output:
i) The restructured tree T'
ii) A pointer to the record i containing k_i
Method:
 1: \tau_j \leftarrow \tau_j + 1
 2: if k_i = k_j then
      if is-left-child(j) = TRUE then
 3:
         \Psi_j \leftarrow 2\tau_j - \tau_{jR} - \tau_{P(j)}
 4:
 5:
      else
         \Psi_j \leftarrow 2\tau_j - \tau_{jL} - \tau_{P(j)}
 6:
      end if
 7:
      if \Psi_i > 0 then
 8:
         rotate-upwards(j)
 9:
         recalculate-tau(j)
10:
         recalculate-tau(P(j))
11:
       end if
12:
       return record j
13:
14: else
       if k_i < k_j then
15:
         CONROT-BST( left-child(j), k_i )
16:
17:
       else
18:
         CONROT-BST( right-child(j), k_i )
       end if
19:
20: end if
End Algorithm
```

The first task accomplished by the Algorithm CONROT-BST is the updating of the counter τ for the present node along the path traversed. After that, the next step consists of determining whether or not the

node with the requested key has been found. When this occurs, the quantities defined by Equations (1) and (2) are computed to determine the value of a quantity referred to as Ψ , where:

$$\Psi_j = 2\tau_j - \tau_{jR} - \tau_{P(j)} \tag{1}$$

when j is the left child of its parent, P(j), and

$$\Psi_j = 2\tau_j - \tau_{jL} - \tau_{P(j)} \tag{2}$$

when j is a right descendant of P(j).

When Ψ is less than zero, an upward rotation is performed. The authors of [16] have shown that this single rotation leads to a decrease in the overall WPL of the *entire* tree. This occur in line 9 of the algorithm, in which the method **rotate-upwards** is invoked. The parameter to this method is a pointer to the node j. The method does the necessary operations required to rotate the node upwards, which means that if the node j is the left child of the parent, then this is equivalent to performing a right rotation over P(j), the parent of j. Analogously, when j is the right child of its parent, the parent of j is left-rotated instead. Once the rotation takes place, it is necessary to update the corresponding counters, τ . Fortunately this task only involve the updating of τ_i , for the rotated node, and the counter of its parent, $\tau_{P(i)}$. The last part of the algorithm, namely lines 14–19, deals with the further search for the key, which in this case is achieved recursively.

The reader will observe that all the tasks invoked in the algorithm are performed in constant time, and in the worst case, the recursive call is done from the root down to the leaves, leading to a O(h) running complexity, where h is the height of the tree.

3.2 The TTOSOM with Conditional Rotations (TTOCONROT)

This section concentrates on the details of the integration between the fields of ADS and the SOM, and in particular, the TTOSOM. Although merging ADS and the SOM is relevant to a wide spectrum of DSs, we focus our scope by considering only tree-based structures. More specifically we shall concentrate on the integration of the CONROT-BST heuristic [16] into a TTOSOM [8], both of which were explained in the preceding sections.

We can conceptually distinguish our method, namely, the Tree-based Topology Oriented SOM with Conditional Rotations (TTOCONROT) from its components and properties.

In terms of components, we detect five elements. First of all, the TTOCONROT has a set of neurons, which, like all SOM-based methods, represents the data space in a condensed manner. Secondly, the TTOCONROT possesses a connection between the neurons, where the neighbor of any specific neuron is based on a nearness measure that is tree-based. The third and fourth components involve the migration of the neurons. Similar to the reported families of SOMs, a subset of neurons closest to the winning neuron are moved towards the sample point using a Vector Quantization (VQ) rule. However, unlike the reported families of SOMs, the identity of the neurons that are moved is based on the tree-based proximity and not on

the feature-space proximity. Finally, the TTOCONROT possesses tree-based mutating operations, namely the above-mentioned conditional rotations.

With respect to the properties of the TTOCONROT, we mention the following. First of all, it is adaptive, with regard to the migration of the points. Secondly, it is also adaptive with regard to the identity of the neurons moved. Thirdly, the distribution of the neurons in the feature space mimics the distribution of the sample points. Finally, by virtue of the conditional rotations, the entire tree is optimized with regard to the overall accesses, which is a unique phenomenon (when compared to the reported family of SOMs) as far as we know.

As mentioned in the introductory section, the general dynamic adaptation of SOM lattices reported in the literature considers essentially adding (and in some cases deleting) nodes/edges. However the concept of modifying the underlying structure's *shape* itself has been unrecorded. Our hypothesis is that this is advantageous by means of a repositioning of the *nodes* and the consequent *edges*, as seen when one performs rotations on a BST. In other words, we place our emphasis on the self-arrangement which occurs as a result of restructuring the DS representing the SOM. In this case, as alluded to earlier, the restructuring process is done between the connections of the neurons so as to attain an asymptotically optimal configuration, where nodes that are accessed more frequently will tend to be placed close to the root. We thus obtain a new species of tree-based SOMs which is self-arranged by performing rotations **conditionally**, **locally** and in a **constant number of steps**.

The primary goal of the field of ADS is to have the structure and its elements attain an optimal configuration as the number of iterations increases. Particularly, among the ADSs that use trees as the underlying topology, the common goal is to minimize the overall access cost, and this roughly means that one places the most frequently accessed nodes close to the root, which is also what CONROT-BST moves towards. Although such an adaptation can be made on any SOM paradigm, the CONROT is relevant to a tree structure, and thus to the TTOSOM. This further implies that some specific settings/modifications must be applied to achieve the integration between the two paradigms.

We start by defining a Binary Search Tree SOM (BSTSOM) as a special instantiation of a SOM which uses a BST as the underlying topology. An Adaptive BSTSOM (ABSTSOM) is a further refinement of the BSTSOM which, during the training process, employs a technique that automatically modifies the configuration of the tree. The goal of this adaptation is to facilitate and enhance the search process. This assertion must be viewed from the perspective that for a SOM, neurons that represent areas with a higher density, will be queried more often.

Every ABSTSOM is characterized by the following properties. First, it is **adaptive**, where, by virtue of the BST representation this adaptation is done by means of rotations, rather than by merely deleting or adding nodes. Second, the neural network corresponds to a BST. The goal is that the NN maintains the essential stochastic and topological properties of the SOM.

3.2.1 Neural Distance

As in the case of the TTOSOM [8], the Neural Distance, d_N , between two neurons depends on the number of unweighted connections that separate them in the user-defined tree. It is consequently the number of edges in the shortest path that connects the two given nodes. More explicitly, the distance between two nodes in the tree, is defined as the minimum number of edges required to go from one to the other. In the case of trees, the fact that there is only a *single* path connecting two nodes implies the uniqueness of the shortest path, and permits the efficient calculation of the distance between them by a node traversal algorithm. Note however, that in the case of the TTOSOM, since the tree itself was *static*, the inter-node distances can be pre-computed *a priori*, simplifying the computational process. The situation changes when the tree is dynamically modified as we shall explain below.

The implications of having the tree which describes the SOM to be dynamic, are three-fold. First of all, the siblings of any given node may change at every time instant. Secondly, the parents and ancestors of the node under consideration could also change at every instant. But most importantly, the structure of the tree itself could change, implying that nodes that were neighbors at any time instant may not continue to be neighbors at the next. Indeed, in the extreme case, if a node was migrated to become the root, the fact that it had a parent at a previous time instant is irrelevant at the next. This, of course, changes the entire landscape, rendering the resultant SOM to be unique and distinct from the state-of-the-art. An example will clarify this.

Consider Figure 3, which illustrates the computation of the neural distance for various scenarios. First, in Figure 3a, we present the scenario when the node accessed is B. Observe that the distances are depicted with dotted arrows, with an adjacent numeric index specifying the current distance from node B. In the example, prior to an access, nodes H, C and E are all at a distance of 2 from node B, even though they are at different levels in the tree. The reader should be aware that non-leaf nodes may also be involved in the calculation, as in the case of node H. Figures 3b and 3c show the process when node B is queried, which in turn triggers a rotation of node B upwards. Observe that the rotation itself only requires local modifications, leaving the rest of the tree untouched. For the sake of simplicity and explicitness, unmodified areas of the tree are represented by dashed lines. Finally, Figure 3d depicts the configuration of the tree after the rotation is performed. At this time instant, C and E are both at distance of 3 from B, which means that they have increased their distance to B by unity. Moreover, although node H has changed its position, its distance to B remains unmodified. Clearly, the original distances are not necessarily preserved as a consequence of the rotation.

Generally speaking, there are four regions of the tree that remain unchanged. These are, namely, the portion of the tree above the parent of the node being rotated, the portion of tree rooted at the right child of the node being rotated, the portion of tree rooted at the left child of the node being rotated, and the portion of tree rooted at the sibling of the node being rotated. Even though these four regions remain unmodified, the neural distance in these regions are affected, because the rotation could lead to a modification of the distances to the nodes.

Another consequence of this operation that is worth mentioning is the following: The distance between any two given nodes that belong to the same unmodified region of the tree is preserved after a rotation is performed. The proof of this assertion is obvious, inasmuch as the fact remains that every path between nodes in any unmodified sub-tree remains with the same sub-tree. This property is interesting because it has the potential to accelerate the computation of the respective neural distances.



Figure 3: Example of the neural distance before and after a rotation. In Figure 3a nodes H, C and E are equidistant from B even though they are at different levels in the tree. Figures 3b and 3c show the process of rotating node A upwards. Finally, Figure 3d depicts the state of the tree after the rotation when only C and E are equidistant from B, and their distance to B has increased by unity. On the other hand, although H has changed its position, its distance to B remains the same.

3.2.2 The Bubble of Activity

A concept closely related to the neural distance, is the one referred to as the Bubble of Activity (BoA) which is the subset of nodes within a distance of r away from the node currently examined. Those nodes are in essence those which are to be migrated toward the signal presented to the network. This concept is valid for all SOM-like NNs, and in particular for the TTOSOM. We shall now consider how this bubble is modified in the context of rotations. The concept of the bubble involves the consideration of a quantity, the so-called *radius*, which establishes how big the BoA is, and which therefore has a direct impact on the number of nodes to be considered. The BoA can be formally defined as [8]

$$B(v_i; T, r) = \{ v | d_N(v_i, v; T) \le r \},$$
(3)

where v_i is the node currently being examined, and v is an arbitrary node in the tree T, whose nodes are V. Note that $B(v_i, T, 0) = \{v_i\}$, $B(v_i, T, i) \supseteq B(v_i, T, i-1)$ and $B(v_i, T, |V|) = V$ which generalizes the special case when the tree is a (simple) directed path.

To clarify how the bubble changes in the context of rotations, we first describe the context when the tree is static. As presented in [8], the function TTOSOM_Calculate_Neighborhood (see Algorithm 3) specifies the steps involved in the calculation of the subset of neurons that are part of the neighborhood of the BMU. This computation involves a collection of parameters, including B, the current subset of neurons in the proximity of the neuron being examined, v, the BMU itself, and $r \in IN$ the current radius of the neighborhood. When the function is invoked for the first time, the set B contains only the BMU marked as the current node, and Algorithm 3 TTOSOM_Calculate_Neighborhood(B, v, r)

Input:

i) B, the set of the nodes in the bubble of activity identified so far. ii) v, the node from where the bubble of activity is calculated. iii) r, the current radius of the bubble of activity. **Output:** i) The set of nodes in the bubble of activity. Method: 1: if $r \leq 0$ then return 2: else 3: for all $child \in v.getChildren()$ do 4: if *child* $\notin B$ then 5: $B \leftarrow B + \{child\}$ 6: 7: TTOSOM_Calculate_Neighborhood(B, child, r-1) end if 8: end for 9: parent=v.getParent(); 10: if $parent \neq NULL$ and $parent \notin B$ then 11: 12: $B \leftarrow B + \{parent\}$ TTOSOM_Calculate_Neighborhood(B, parent, r-1) 13:end if 14:15: end if End Algorithm

through a recursive call, B will end up storing the entire set of units within a radius r of the BMU. The tree is recursively traversed for all the direct topological neighbors of the current node, i.e., in the direction of the direct parent and children. Every time a new neuron is identified as part of the neighborhood, it is added to B and a recursive call is made with the radius decremented by one unit⁶, marking the recently added neuron as the current node.

The question of whether or not a neuron should be part of the current bubble, depends on the number of connections that separate the nodes rather than the distance that separate the networks in the solution space (for instance, the Euclidean distance). Figure 4 depicts how the BoA differs from the one defined by the TTOSOM as a result of applying a rotation. Figure 4a shows the BoA around the node B, using the same configuration of the tree as in Figure 3a, i.e., before the rotation takes place. Here, the BoA when r = 1involves the nodes $\{B, A, D, F\}$, and when r = 2 the nodes contained in the bubble are $\{B, A, D, F, C, E, H\}$. Subsequently, considering a radius equal to 3, the resulting BoA contains the nodes $\{B, A, D, F, C, E, H, G, I\}$. Finally, the r = 5 case leads to a BoA which includes the whole set of nodes. Now, observe the case presented in Figure 4b, which corresponds to the BoA around B after the rotation upwards has been effected, i.e., the same configuration of the tree used in Figure 3d. In this case, when the radius is unity, nodes $\{B, A, F\}$ are the only nodes within the bubble, which is different from the corresponding bubble before the rotation is invoked. Similarly, when r = 2, we obtain a set different from the analogous pre-rotation case, which in this case is $\{B, A, F, D, H\}$. Note that coincidentally, for the case of a radius equal to 3, the bubbles are identical before and after the rotation, i.e., they invoke the nodes $\{B, A, D, F, G, I\}$. Trivially, again, when r = 5, the BoA invokes the entire tree.

⁶This fact will ensure that the algorithm reaches the base case when r = 0.



Figure 4: The BoA associated with the TTOSOM before and after a rotation is invoked at node B.

As explained, Equation (3) describes the criteria for a BoA calculated on a *static* tree. It happens that, as a result of the conditional rotations, the tree will be dynamically adapted, and so the entire phenomenon has to be re-visited. Consequently, the BoA around a particular node becomes a function of time, and, to reflect this fact, Equation (3) should be reformulated as:

$$B(v_i; T, r, t) = \{ v | d_N(v_i, v; T, t) \le r \},$$
(4)

where t is the discrete time index.

The algorithm to obtain the BoA for a specific node in such a setting is identical to Algorithm 3, except that the input tree itself dynamically changes. Further, even though the formal notation includes the time parameter, "t", it happens that, in practice, the latter is needed only if the user/application requires a history of the BoA for any or all the nodes. Storing the history of BoAs will require the maintenance of a DS that will primarily store the changes made to the tree itself. Although storing the history of changes made to the tree can be done optimally [31], the question of explicitly storing the entire history of the BoAs for all the nodes in the tree remains open.

3.2.3 Enforcing the BST Property

The CONROT-BST heuristic [16] requires that the tree should possess the BST property [18]:

Let x be a node in a BST. If y is a node in the left subtree of x, then $key[y] \le key[x]$. Further, if y is a node in the right subtree of x, then $key[x] \le key[y]$.

To satisfy the BST property, first of all we see that, the tree must be binary⁷. As a general TTOSOM utilizes an arbitrary number of children per node, one possibility is to bound the value of the branching factor to be 2. In other words, the tree trained by the TTOSOM is restricted to contain at most two children per node. Additionally, the tree must implicitly involve a comparison operator between the two children so as to discern between the branches and thus perform the search process. This comparison can be achieved by defining a unique key that must be maintained for each node in the tree, and which will, in turn, allow a

⁷Of course, this is a severe constraint. But we are forced to require this, because the phenomenon of achieving conditional rotations for arbitrary k-ary trees is unsolved. This research, however, is currently being undertaken.

lexicographical arrangement of the nodes.

This leads to a different, but closely related concept, which concerns the preservation of the topology of the SOM. During the training process, the configuration of the tree will change as the tree evolves, positioning nodes that are accessed more often closer to the root. This probability-based ordering, will hopefully, be preserved by the rotations.

A particularly interesting case occurs when the imposed tree corresponds to a *list* of neurons, i.e., a 1-ary tree. If the TTOSOM is trained using such a tree where each node has at most two children, then the adaptive process will alter the original list. The rotations will then modify the original configuration, generating a new state, where the non-leaf nodes might have one or two children each. In this case the consequence of incorporating ADS-based enhancements to the TTOSOM will imply that the results obtained will be significantly different from those shown in [8].

As shown in [35], an optimal arrangement of the nodes of the tree can be obtained using the probabilities of accesses. If these probabilities are not known *a priori*, then the CONROT-BST heuristic offers a solution, which involves a decision of whether or not to perform a single rotation towards the root. It happens that the concept of the "just accessed" node in the CONROT-BST is compatible with the corresponding BMU defined for the CL model. In CL, a neuron may be accessed more often than others and some techniques take advantage of this phenomenon through the inclusion of strategies that add or delete nodes

The CONROT-BST implicitly stores the information acquired by the currently accessed node by incrementing a counter for that node. This is (in a distant sense) akin to the concept of a BMU counter which adds or delete nodes in competitive networks.

During the training phase, when a neuron is a frequent winner of the CL, it gains prominence in the sense that it can represent more points from the original data set. This phenomenon is registered by increasing the BMU counter for that neuron. We propose that during the training phase, we can verify if it is worth modifying the configuration of the tree by moving this neuron one level up towards the root as per the CONROT-BST algorithm, and consequently explicitly recording the relevant role of the particular node with respect to its nearby neurons. CONROT-BST achieves this by performing a *local* movement of the node, where only its direct parent and children are aware of the neuron promotion.

Neural Promotion is the process by which a neuron is relocated in a more privileged position⁸ in the network with respect to the other neurons in the NN. Thus, while all "all neurons are born equal", their importance in the society of neurons is determined by what they represent. This is achieved, by an explicit advancement of its rank or position. Given this premise, the nodes in the tree will be adapted in such a way that neurons that have been BMUs more frequently, will tend to move towards the root if an only if a reduction in the overall WPL is obtained as a consequence of such a promotion. The properties of CONROT-BST guarantee this.

Once the SOM and BST are "tied" together in a symbiotic manner (where one enhances the other and vice versa), the adaptation can be achieved by affecting the configuration of the BST. This task will be performed every time a training step of the SOM is performed. Clearly, it is our task to achieve an integration of the

⁸As far as we know, we are not aware of any research which deals with the issue of Neural Promotion. Thus, we believe that this concept, itself, is pioneering.

BST and the SOM, and Figure 5 depicts the main architecture used to accomplish this. It transforms the structure of the SOM by modifying the configuration of the BST that, in turn, holds the structure of the neurons.



Figure 5: Architectural view of an Adaptive Tree-Based SOM.

As this work constitutes the first attempt to constraint a tree-based SOM using a BST, our focus is placed on the self-adaptation of the nodes. In this sense, the unique identifiers of the nodes are employed to maintain the BST structure and to promote nodes that are frequently accessed towards the root. We are currently examining ways to enhance this technique so as to improve the time required to identify the BMU as well.

3.2.4 Initialization

Initialization, in the case of the BST-based TTOSOM, is accomplished in two main steps which involve defining the initial value of each neuron and the connections among them. The initialization of the codebook vectors are performed in the same manner as in the basic TTOSOM. The neurons can assume a starting value arbitrarily, for instance, by placing them on randomly selected input samples. On the other hand, a major enhancement with respect to the basic TTOSOM lays in the way the neurons are linked together. The basic definition of the TTOSOM utilizes connections that remain static through time. The beauty of such an arrangement is that it is capable of reflecting the user's perspective at the time of describing the topology, and it is able to preserve this configuration until the algorithm reaches convergence. The inclusion of the rotations renders this dynamic.

3.2.5 The Required Local Information

In our proposed approach, the codebooks of the SOM correspond to the nodes of a BST. Apart from the information regarding the codebooks themselves in the feature space, each neuron requires the maintenance of additional fields to achieve the adaptation. Besides this, each node inherits the properties of a BST Node, and it thus includes a pointer to the left and right children, as well as (to make the implementation easier), a pointer to its parent. Each node also contains a label which is able to uniquely identify the neuron when it is in the "company" of other neurons. This identification index constitutes the lexicographical key used to sort the nodes of the tree and remains static as time proceeds. Figure 6 depicts all the fields included in a

neuron of a BST-based SOM.



Figure 6: Fields included in a BST-based SOM neuron.

3.2.6 The Neural State

The different states that a neuron may assume during its lifetime are illustrated in Figure 7. At first, when the node is created, it is assigned a unique identifier, and the rest of the data fields are populated with their initial values. Here, the codebook vector assumes a starting value in the feature space, and the pointers are configured so as to appropriately link the neuron with the rest of the neurons in the tree in a BST configuration. Next, during the most significant portion of the algorithm, the NN enters a main loop, where training is effected. This training phase, involves adjusting the codebooks and may also trigger optional modules that affect the neuron. Once the BMU is identified, the neuron might assume the "restructured" state, which means that a restructuring technique, such as the CONROT algorithm, will be applied. Alternatively, the neuron might be ready to accept queries, i.e., be part of the CL process in the mapping mode. Additionally, an option that we are currently investigating, involves the case when a neuron is no longer necessary and may thus be eliminated from the main neural structure. We refer to this state as the so-called "deleted" state, and it is depicted using dashed lines. Finally, we foresee an alternative state referred to as the "frozen" state, in which the neuron does not participate in the CL during the training mode although it may continue to be part of the overall NN structure.



Figure 7: Possible states that a neuron may assume.

3.2.7 The Training Step of the TTOCONROT

The training module of the TTOCONROT is responsible of determining the BMU, performing restructuring, calculating the BoA and migrating the neurons within the BoA. Basically, what has to be done, is to integrate the CONROT algorithm into the sequence of steps responsible for the training phase of the TTOSOM.

Algorithm 4 describes the details of how this integration is accomplished. Line 1 performs the first task of the algorithm, which involves determining the BMU. After that, line 2 invokes the CONROT procedure. The rationale for following this sequence of steps is that the parameters needed to perform the conditional rotation, as specified in [16], includes the "key" of the element queried, which, in the present context, corresponds to the identity of the BMU. At this stage of the algorithm, the BMU may be rotated or not depending on the optimizing criterion given by equations (1) and (2), and the BoA is determined *after* this restructuring is done. These are performed in lines 3 and 4 of the algorithm respectively. Finally, lines 5 to 7, are responsible for the neural migration itself, and oversee the movement of the neurons within the BoA towards the input sample.

```
Algorithm 4 TTOCONROT-BST_train(x,p)
```

```
Input:

i) x, a sample signal.

ii) p, the pointer to the tree.

Method:

1: v \leftarrow TTOSOM\_Find\_BMU(x,p)

2: cond=rot=bst(p,v.getID())

3: B \leftarrow \{v\}

4: TTOSOM\_Calculate\_Neighborhood(B,v,radius)

5: for all b \in B do

6: update\_rule(b.getCodebook(),x)

7: end for

End Algorithm
```

3.2.8 Alternative Restructuring Techniques

Even though, we have explained the advantages of the CONROT algorithm, the architecture that we are proposing allows the inclusion of alternative restructuring modules other than the CONROT. Potential candidates which can be used to perform the adaptation are the ones mentioned in Section 3.1 and include the splay and the MT algorithms, among others.

4 Experimental Results

To *illustrate* the capabilities of our method, the experiments reported in the present work are primarily focused in lower dimensional feature spaces. This will help the reader in geometrically visualizing the results we have obtained. However, it is important to remark that the algorithm is also capable of solving problems in higher dimensions, although a graphical representation of the results will not be as illustrative. We know that, as per the results obtained in [8], the TTOSOM is capable of inferring the distribution and structure of the data. However, in this present setting, we are interested in investigating the effects of applying the neural rotation as part of the training process. To render the results comparable, the experiments in this section use the same schedule for the learning rate and radius, i.e., no particular refinement of the parameters has been done for any specific data set. Additionally, the parameters follow a rather "slow" decrement of the so-called decay parameters, allowing us to understand how the prototype vectors are moved as convergence

takes place. When solving practical problems, we recommend a further refinement of the parameters so as to increase the speed of the convergence process.

4.1 **TTOCONROT's Structure Learning Capabilities**

We shall describe the performance of TTOCONROT with data sets in 1, 2 and 3 dimensions, as well as experiments in the multidimensional domain. The specific advantages of the algorithm for various scenarios will also be highlighted.

4.1.1 One Dimensional Objects

Since our entire learning paradigm assumes that the data has a tree-shaped model, our first attempt was to see how the philosophy is relevant to a unidimensional object (i.e., a curve), which really possesses a "linear" topology. Thus, as a prima facie case, we tested the strength of the TTOCONROT to infer the properties of data sets generated from linear functions in the plane. Figure 8 shows different snapshots of how the TTOCONROT learns the data generated from a curve. Random initialization was used by uniformly drawing points from the unit square. Observe that the original data points do not lie in the curve. Our aim here was to show how our algorithm could learn the structure of the data using arbitrary (initial and "non-realistic") values for the codebook vectors. Figures 8b and 8c depict the middle phase of the training process, where the edges connecting the neurons are omitted for simplicity. It is interesting to see how, after a few hundred training steps, the original chaotic placement of the neurons are rearranged so as to fall within the line described by the data points. The final configuration is shown in Figure 8d. The reader should observe that after convergence has been achieved, the neurons are placed almost equidistantly along the curve. Even though the codebooks are not sorted in and increasing numerical order, the hidden tree and its root, denoted by two concentric squares, are configured in such a way that nodes that are queried more frequently will tend to be closer to the root. In this sense, the algorithm is not only capturing the essence of the topological properties of the data set, but at the same time rearranging the internal order of the neurons according to their importance in terms of their probabilities of access.

4.1.2 Two Dimensional Data Points

To demonstrate the power of including ADS in SOMs, we shall now consider the same two-dimensional data sets studied in [8]. First we consider the data generated from a triangular-spaced distribution, as shown in Figures 9a-9d. In this case, the initial tree topology is unidirectional, i.e., a list, although, realistically, this is quite inadvisable considering the true (unknown) topology of the distribution. In other words, we assume that the user has *no a priori* information about the data distribution. Thus, for the initialization phase, a 1-ary tree is employed as the tree structure, and the respective keys are assigned in an increasing order. Observe that in this way we are providing minimal information to the algorithm. The root of the tree is marked with two concentric squares, i.e., the neuron labeled with the index 0 in Figure 9a. Also, with regards to the feature space, the prototype vectors are initially randomly placed. In the first iteration, the linear topology is lost, which is attributable to the randomness of the data points. As the prototypes are migrated



Figure 8: A 1-ary tree, i.e., a list topology, learns a curve. For the sake of simplicity, the edges are ommitted.

and reallocated (see Figures 9b and 9c), the 1-ary tree is modified as a consequence of the *rotations*. Such a transformation is completely novel to the field of SOMs. Finally, Figure 9d depicts the case after convergence has taken place. Here, the tree nodes are uniformly distributed over the entire triangular domain. The BST property is still preserved, and further rotations are still possible if the training process continues.

This experiment serves as an excellent example to show the differences between our current method and the original TTOSOM algorithm [8], where the same data set with similar settings was utilized. In the case of the TTOCONROT the points effectively represent the entire data set. However, the reader must observe that we do not have to provide the algorithm with any particular *a priori* information about the structure of the data distribution – this is learned during the training process, as shown in Figure 9d. Thus, the specification of the initial "user-defined" tree topology (representing his perspective of the data space) required by the TTOSOM is no longer mandatory, and an alternative specification which *only* requires the number of nodes in the initial 1-ary tree is sufficient.

A second experiment involves a Gaussian distribution. Here a 2-dimensional Gaussian ellipsoid is learned using the TTOCONROT algorithm. The convergence of the entire training execution phase is displayed in Figure 10. This experiment considers a complete BST of depth 4, i.e., containing 15 nodes. For simplicity the labels of the nodes have been removed.

In Figure 10, the tree structure generated by the neurons suggest an ellipsoidal structure for the data distribution. This experiment is a good example to show how the nodes close to the root represent dense areas of the ellipsoid, and at the same time, those node that are far from the root (in tree space) occupy regions with low density, e.g., in the "extremes" of the ellipse. The TTOCONROT infers this structure without receiving any *a priori* information about the distribution or its structure.

The experiment shown in Figures 11a-11d considers data generated from an irregular shape with a concave surface. Again, as in the case of the experiments described earlier, the original tree includes 15 neurons arranged unidirectionally, i.e., as in a list. As a result of the training, the distribution is learned and the



Figure 9: A 1-ary tree, i.e., a *list* topology, learns a triangular distribution. The DS is self-adapted so that nodes accessed more frequently are moved closer to the root conditionally. The BST property is also preserved.



Figure 10: A tree learns a Gaussian distribution. The neurons that are accessed more frequently are promoted closer to the root.

tree is adapted accordingly, as illustrated in Figure 11d. Observe that the random initialization is performed by randomly selecting points from the unit square, and this points thus do not necessarily fall within the concave boundaries. Although this initialization scheme is responsible of placing codebook vectors outside of the irregular shape, the reader should observe that in a few training steps, they are repositioned inside the contour. It is important to indicate that, even though after the convergence of the algorithm, a line connecting two points passes outside the overall "unknown" shape, one must take into account that the TTOCONROT tree attempts to mimic the stochastic properties in terms of access probabilities. When the user desires the topological mimicry in terms of skeletal structure, we recommend the use of the TTOSOM instead. The final distribution of the points is quite amazing!



Figure 11: A 1-ary tree, i.e., a list topology, learns different distributions from a concave object using the TTOCONROT algorithm. The set of parameters is the same as in the other examples.

4.1.3 Three Dimensional Data Points

We will now explain the results obtained when applying the algorithm with and without CONROT. To do this we opt to consider three-dimensional objects. The experiments utilize the data generated from the contour of the unit sphere. It also initially involves an *uni*-dimensional chain of 31 neurons. Additionally, in order to show the power of the algorithm, both cases initialize the codebooks by randomly drawing points from the unit cube, which thus initially places the points outside the sphere itself. Figure 12 presents the case when the basic TTO algorithm (without CONROT) learns the unit sphere without performing conditional rotations. The illustration presented in Figure 12a show the state of the neurons before the first iteration is completed. Here, as shown, the codebooks lie inside the unit cube, although some of the neurons are positioned outside the boundary of the respective circumscribed sphere, which is the one we want to learn. Secondly, Figures 12b and 12c depict intermediate steps of the learning phase. As the algorithm processes the information provided by the sample points and the neurons are repositioned, the chain of neurons is constantly "twisted" so as to adequately represent the entire manifold. Finally, Figure 12d illustrates the case when the convergence is reached. In this case, the one-dimensional list of neurons is evenly distributed over the sphere, preserving the original properties of the 3-dimensional object and also presenting a shape which reminds the viewer of the so-called Peano curve [47].

A complimentary set of experiments which involved the learning of the same unit sphere where the TTO scheme was augmented by conditional rotations (i.e., CONROT) was also conducted. Figure 13a shows the initialization of the codebooks. Here, the starting positions of the neurons fall within the unit cube as in the case displayed in Figure 12a. Figures 13b and 13c show snapshots after 1,000 and 3,000 iterations respectively. In this case the tree configuration obtained in the intermediate phases differ significantly from those obtained by the corresponding configurations shown in Figure 12, i.e., those that involved no rotations. In this case, the list rearranges itself as per CONROT, modifying the original chain structure to yield a more-



(a) After 0 iterations (b) After 1,000 iter. (c) After 3,000 iter. (d) After 5,000 iter.

Figure 12: A 1-ary tree, i.e., a list topology, learns a sphere distribution when the algorithm does not utilize any conditional rotation.

or-less balanced tree. Finally, from the results obtained after convergence, and illustrated in Figure 13d, it is possible to compare both scenarios. In both cases, we see that the tree is accurately learned. However, in the first case, the structure of the nodes is maintained as a list throughout the learning phase, while, in the case when CONROT is applied, the configuration of the tree is constantly revised, promoting those neurons that are queried more frequently. Additionally, the experiments show us how the dimensionality reduction property evidenced in the traditional SOM, is also present in the TTOCONROT. Here, an object in the 3-dimensional domain is successfully learned by our algorithm, and the properties of the original manifold are captured from the perspective of a tree.



(a) After 0 iterations(b) After 1,000 iter.(c) After 3,000 iter.(d) After 5,000 iter.Figure 13: A 1-ary tree, i.e., a list topology, learns a sphere distribution.

4.1.4 Multidimensional Data Points

The well known Iris dataset was chosen for showing the power of our scheme in a scenario when the dimensionality is increased. This data set gives the measurements (in centimeters) of the variables which are the sepal length, sepal width, petal length and petal width, respectively, for 50 flowers from each of 3 species of the iris family. The species are the Iris Setosa, Versicolor, and Virginica.

In this set of experiments, the Iris data set was learned under three different configurations, using a fixed schedule for the learning rate and radius but with a distinct tree configuration. The results of the experiments are depicted in Figure 14 and involve a complete binary tree of depth 3, 4 and 5, respectively. Taking into

account that the dataset possesses a high dimensionality, we present the projection in the 2-dimensional space to facilitate the visualization. We also removed the labels from the nodes in Figure 14c to improve understandability.



Figure 14: Three different experiments where the TTOCONROT effectively captures the fundamental structure of the iris dataset (2-dimensional projection of the data is shown).

The experiment utilizes a underlying tree topology of a complete binary tree with different levels of depth. By this we attempt to show examples of how exactly the *same* parameters of the TTOCONROT, can be utilized to learn the structure from data belonging to the 2-dimensional, 3-dimensional and also 4-dimensional spaces. After executing the TTO-SOM, each of the main branches of the tree were migrated towards the center of mass of the cloud of points in the hyper-space belonging to each of the three categories of flowers, respectively.

Since the TTOCONROT is an unsupervised learning algorithm, it performs learning without knowing the true labels of the samples. However, when these labels are available, one can use them to evaluate the quality of the tree. To do so, each sample is assigned to its closest neuron, and tagging the neuron with the class which is most frequent. Table 1 presents the evaluation for the tree in Figure 14a.

Assigned to neuron \rightarrow	1	2	3	4	5	6	7
Iris-setosa	0	0	0	0	0	50	0
Iris-versicolor	0	0	20	7	20	0	3
Iris-virginica	12	22	0	0	1	0	15

Table 1: "Cluster to class" evaluation for the tree in Figure 14a.

Using the simple voting scheme explained above, it is possible to see from Table 1, that only 4 instances are incorrectly classified, i.e., 97.3% of the instances are correctly classified. Additionally, observe that node 6 contains all the 50 instances corresponding to the class Iris-setosa. It is well known that the Iris-setosa class is linearly separable from the other two classes, and our algorithm was able to discover this without providing it with the labels. We find this result quite fascinating!

The experimental results shown in Table 1, not only demonstrate the potential capabilities of the TTOCONROT for performing clustering, but also suggest the possibilities of using it for pattern classification. According to [23], there are several reasons for performing pattern classification using an unsupervised approach. We are currently investigating such a classification strategy.

4.2 Skeletonization

In general, the main objective of skeletonization consists of generating a simpler representation of the shape of an object. The authors of [44] refer to skeletonization in the plane as the process by which a 2-dimensional shape is transformed into a 1-dimensional one, similar to a "stick" figure. The applications of skeletonization are diverse, including the fields of Computer Vision and Pattern Recognition.

As explained in [8], the traditional methods for skeletonization assume the connectivity of the data points and when this is not the case, more sophisticated methods are required. Previous efforts involving SOM variants to achieve skeletonization has been proposed [8, 19, 53]. We remark that the TTOSOM [8] is the only one which uses a tree-based structure. The TTOSOM assumed that the shape of the object is not known *a priori*. Rather, this is learned by accessing a single point of the entire shape at any time instant. Our results reported in [8] confirm that this is actually possible, and we now focus on how the conditional rotations will affect such a skeletonization.

Figure 15 shows how the TTOCONROT learned the skeleton of different objects in the 2-dimensional and the 3-dimensional domain. In all the cases the same schedule of parameters were used, and only the number of neurons employed was chosen proportionally to the number of data points contained in the respective data sets. It is important to remark that we did not invoke any post-processing of the edges, e.g., minimum spanning tree, and that the skeleton observed was exactly what our BSTSOM learned. Firstly, Figures 15a-15d illustrate the shapes of the silhouette of a human, a rhinoceros, a 3d representation of a head, and a 3d representation of a woman. The figures also show the trees learned from the respective data sets. Additionally figures 15e-15h display only the data points, which in our opinion are capable of representing the fundamental structure of the four objects in a 1-dimensional way effectively.

As a final comment, we stress that all the shapes employed in the experiments involve the learning of the "external" structure of the objects. For the case of solid objects, if the internal data points are also provided, the TTOCONROT is able to give an approximation of the so-called endo-skeleton, i.e., a representation in which the skeleton is built inside the solid object.

4.3 Theoretical Analysis

According to Kiviluoto [34], there are three different criteria for evaluating the quality of a map. The first criterion indicates how *continuous* the mapping is, implying that input signals that are close (in the input space) should be mapped to codebooks that are close in the output space as well. A second criterion involves the *resolution* of the mapping. Maps with high resolution possess the additional property that input signals that are distant in the input space should be represented by distant codebooks in the output space. A third criterion imposed on the accuracy of the mapping is aimed to reflect the *probability distribution* of the input set. There exist a variety of measures for quantifying the quality of maps, and these include the Quantization error, the Topographic Product [11], the Topographic Error [34] and the Trustworthiness and Neighborhood Preservation [55]. Although we are currently investigating [6] how the quality of any tree-based SOM (not just our scheme) can be quantified using these metrics. The following arguments are pertinent.



Figure 15: TTOCONROT effectively captures the fundamental structure of four objects in a 1-dimensional way. Figures 15a-15d show the silhouette of a human, a rhinoceros, a 3d representation of a head, and a 3d representation of a woman, as well as the respective trees learned. Figures 15e-15h show only the respective data points.

The ordering of the weights (with respect to the position) of the neurons of the SOM has been proved for unidimensional topologies [17, 36, 51]. Extending these results to higher dimensional configurations or topologies leads to numerous unresolved problems. First of all, the question of what one means by "ordering" in higher dimensional spaces has to be defined. Further, the issue of the "absorbing" nature of the "ordered state" is open. Budinich, in [14], explains intuitively the problems related to the ordering of neurons in higher dimensional configurations. Huang *et al.* [29] introduce a definition of the ordering and show that even though the position of the codebook vectors of the SOM have been ordered, there is still the possibility that a sequence of stimuli will cause their disarrangement. Some statistical indexes of correlation between the measures of the weights and distances of the related positions have been introduced in [10].

With regard to the topographic product, the authors of [11] have shown the power of the metric by applying it on different artificial and real-world data sets, and also compared it with different measures to quantify the topology [10]. Their study concentrates on the traditional SOM, implying that the topologies evaluated were of a "linear" nature, with the consequential extension to 2-dimensions and 3-dimensions by means of grids only. In [28], Haykin mention that the Topographic Product may be employed to compare the quality of different maps, even when these maps possess different dimensionality. However, he also noted that this measurement is only possible when the dimensionality of the topological structure is the same as the dimensionality of the feature space. Further, tree-like topologies were not considered in their study. To be more precise, most of the effort towards determining the concept of topology preservation for dimensions greater than unity are specifically focused on the SOM [11, 17, 36, 14, 29, 10], and do not define how a treelike topology should be measured nor how to define the order in topologies which are not grid-based. Thus, we believe that even the tools to analyze the TTOCONROT are currently not available. The experimental results obtained in our paper, suggest that the TTOCONROT is able to train the NN so as to preserve the stimuli. However, in order to quantify the quality of this topology, the matter of defining a concept of ordering on tree-based structure has yet to be resolved. Although this issue is of great interest to us, this rather ambitious task lies beyond the scope of our present manuscript.

5 Conclusions and Discussions

5.1 Concluding Remarks

In this paper, we have proposed a novel integration between the areas of Adaptive Data Structures (ADSs) and the Self-Organizing Maps (SOMs). In particular we have shown how a tree-based SOM can be adaptively transformed by the employment of an underlying Binary Search Tree (BST) structure and subsequently, restructured using rotations that are performed conditionally. These rotations on the nodes of the tree are local, can be done in constant time, and performed so as to decrease the Weighted Path Length (WPL) of the entire tree. One of the main advantages of the algorithm, is that the user does not need to have a priori knowledge about the topology of the input data set. Instead, our proposed method, namely the TTOSOM with Conditional Rotations (TTOCONROT), infers the topological properties of the stochastic distribution, and at the same time, attempt to build the best BST that represents the data set.

Incorporating the data structure's constraints in this ways has not being achieved by any of the related approaches included in the state-of-the-art. Our premise is that regions of the hyper-space that are accessed more often should be promoted to preferential spots in the tree representation, which yields to an improved stochastic representation.

As our experimental results suggest, the TTOCONROT tree is indeed able to absorb the stochastic properties of the input manifold. It is also possible to obtain a tree configuration that can learn both, the stochastic properties in terms of access probabilities *and* at the same time preserve the topological properties in terms of its skeletal structure.

5.2 Discussions and Future Work

As explained in Section 4.3, the work associated with measuring the topology preservation of the SOM, including the proof of its convergence for the unidimensional case, has been performed for the traditional SOM only. The questions are unanswered for how a tree-like topology should be measured, and for defining the order in topologies which are not grid-based. Thus, we believe that even the tools for formally analyzing the TTOCONROT are currently not available. The experimental results obtained in our paper, suggest that the TTOCONROT is able to train the Neural Network (NN) so as to preserve the stimuli for which the concept of ordering on tree-based structures has yet to be resolved.

Even though our principal goal was to obtain a more accurate representation of the stochastic distribution, our results also suggest that the special configuration of the tree obtained by the TTOCONROT can be further exploited so as to improve the time required for identifying the Best Matching Unit (BMU). The state-ofthe-art includes different strategies that expand trees by inserting nodes (which can be a single neuron or a SOM-layer) that essentially are based on a Quantization Error (QE) measure. In some of these strategies, the error measure is based on the "hits", i.e., the number of times a neuron has been selected as the BMU, which is, in principle, the same type of counter utilized by the Conditional Rotations (CONROT). Our strategy, TTOCONROT, which asymptotically positions frequently accessed nodes close to the root, might incorporate a module, that taking advantage of the "optimal" tree and the BMU counters already present in the TTOCONROT, splits the node at the root level. Thus, the splitting operation will occur without the necessity of searching for the node with the largest QE, under the assumption that a higher number of hits indicates that the degree of granularity of a particular neuron is lacking refinement. The concept of using the root of the tree for growing a tree-based SOM is itself pioneering, as far as we know, and the design and implementation details of this are currently being investigated.

References

- M. Adelson-Velskii and M. E. Landis. An algorithm for the organization of information. Sov. Math. DokL, 3:1259–1262, 1962.
- [2] M. U. Akram, S. Khalid, and S. A. Khan. Identification and classification of microaneurysms for early detection of diabetic retinopathy. *Pattern Recognition*, 46(1):107–116, 2013.
- [3] D. Alahakoon, S. K. Halgamuge, and B. Srinivasan. Dynamic self-organizing maps with controlled growth for knowledge discovery. *IEEE Transactions on Neural Networks*, 11(3):601–614, 2000.
- [4] B. Allen and I. Munro. Self-organizing binary search trees. J. ACM, 25(4):526–535, 1978.
- [5] E. Arsuaga Uriarte and F. Díaz Martín. Topology preservation in SOM. International Journal of Applied Mathematics and Computer Sciences, 1(1):19–22, 2005.
- [6] C. A. Astudillo. Self Organizing Maps Constrained by Data Structures. PhD thesis, Carleton University, 2011.
- [7] C. A. Astudillo and B. J. Oommen. On using adaptive binary search trees to enhance self organizing maps. In A. Nicholson and X. Li, editors, 22nd Australasian Joint Conference on Artificial Intelligence (AI 2009), pages 199–209, 2009.
- [8] C. A. Astudillo and B. J. Oommen. Imposing tree-based topologies onto self organizing maps. *Informa*tion Sciences, 181(18):3798–3815, 2011.
- C. A. Astudillo and B. J. Oommen. On achieving semi-supervised pattern recognition by utilizing tree-based SOMs. *Pattern Recognition*, 46(1):293 – 304, 2013.
- [10] H. U. Bauer, M. Herrmann, and T. Villmann. Neural maps and topographic vector quantization. Neural Networks, 12(4-5):659 – 676, 1999.
- [11] H. U. Bauer and K. R. Pawelzik. Quantifying the neighborhood preservation of self-organizing feature maps. *Neural Networks*, 3(4):570–579, July 1992.

- [12] J. R. Bitner. Heuristics that dynamically organize data structures. SIAM J. Comput., 8:82–110, 1979.
- [13] J. Blackmore. Visualizing high-dimensional structure with the incremental grid growing neural network. Master's thesis, University of Texas at Austin, 1995.
- [14] M. Budinich. On the ordering conditions for self-organizing maps. Neural Computation, 7(2):284–289, 1995.
- [15] G. A. Carpenter and S. Grossberg. The art of adaptive pattern recognition by a self-organizing neural network. *Computer*, 21(3):77–88, 1988.
- [16] R. P. Cheetham, B. J. Oommen, and D. T. H. Ng. Adaptive structuring of binary search trees using conditional rotations. *IEEE Trans. on Knowl. and Data Eng.*, 5(4):695–704, 1993.
- [17] P. L. Conti and L. De Giovanni. On the mathematical treatment of self organization: extension of some classical results. In Artificial Neural Networks - ICANN 1991, International Conference, volume 2, pages 1089–1812, 1991.
- [18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms, Second Edition. McGraw-Hill Science/Engineering/Math, July 2001.
- [19] A. Datta, S. M. Parui, and B. B. Chaudhuri. Skeletal shape extraction from dot patterns by selforganization. *Pattern Recognition*, 1996., Proceedings of the 13th International Conference on, 4:80–84 vol.4, Aug 1996.
- [20] D. Deng. Content-based image collection summarization and comparison using self-organizing maps. Pattern Recognition, 40(2):718 – 727, 2007.
- [21] M. Dittenbach, D. Merkl, and A. Rauber. The growing hierarchical self-organizing map. In Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on, volume 6, pages 15–19 vol.6, 2000.
- [22] J. Dopazo and J. M. Carazo. Phylogenetic reconstruction using an unsupervised growing neural network that adopts the topology of a phylogenetic tree. *Journal of Molecular Evolution*, 44(2):226–233, February 1997.
- [23] R. Duda, P. E. Hart, and D. G. Stork. Pattern Classification (2nd Edition). Wiley-Interscience, 2000.
- [24] B. Fritzke. Growing Cell Structures a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460, 1994.
- [25] B. Fritzke. Growing Grid a self-organizing network with constant neighborhood range and adaptation strength. Neural Processing Letters, 2(5):9–13, 1995.
- [26] B. Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, Advances in Neural Information Processing Systems 7, pages 625–632, Cambridge MA, 1995. MIT Press.

- [27] L. Guan. Self-organizing trees and forests: A powerful tool in pattern clustering and recognition. In Image Analysis and Recognition, Third International Conference, ICIAR 2006, Póvoa de Varzim, Portugal, September 18-20, 2006, Proceedings, Part I, pages I: 1–14, 2006.
- [28] S. Haykin. Neural Networks and Learning Machines. Prentice Hall, 3rd edition edition, 2008.
- [29] G. Huang, H. A. Babri, and H. Li. Ordering of self-organizing maps in multi-dimensional cases. Neural Computation, 10:19–24, 1998.
- [30] H.-G. Kang and D. Kim. Real-time multiple people tracking using competitive condensation. Pattern Recognition, 38(7):1045 – 1058, 2005.
- [31] H. Kaplan. Handbook of Data Structures and Applications, chapter 31: Persistent Data Structures, pages 31.1 31.26. Chapman and Hall/CRC, 2004.
- [32] S. Kaski, J. Kangas, and T. Kohonen. Bibliography of self-organizing map (SOM) papers: 1981–1997. Neural Computing Surveys, 1:102–350, 1998.
- [33] M. H. Khosravi and R. Safabakhsh. Human eye sclera detection and tracking using a modified timeadaptive self-organizing map. *Pattern Recognition*, 41(8):2571–2593, 2008.
- [34] K. Kiviluoto. Topology preservation in self-organizing maps. In P. IEEE Neural Networks Council, editor, Proceedings of International Conference on Neural Networks, ICNN'96, volume 1, pages 294–299, New Jersey, USA, 1996.
- [35] D. E. Knuth. The art of computer programming, volume 3: (2nd ed.) sorting and searching. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.
- [36] T. Kohonen. Self-Organizing Maps. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1995.
- [37] P. Koikkalainen and E. Oja. Self-organizing hierarchical feature maps. IJCNN International Joint Conference on Neural Networks, 2:279–284, June 1990.
- [38] T. W. H. Lai. Efficient maintenance of binary search trees. PhD thesis, University of Waterloo, Waterloo, Ont., Canada, 1990.
- [39] Y. Liang, M. C. Fairhurst, and R. M. Guest. No titlea synthesised word approach to word retrieval in handwritten documents. *Pattern Recognition*, 45(12):4225–4236, 2012.
- [40] M. Martinetz and K. J. Schulten. A "neural-gas" network learns topologies. In in Proceedings of International Conference on Articial Neural Networks, volume I, pages 397–402, North-Holland, Amsterdam, 1991.
- [41] K. Mehlhorn. Dynamic binary search. SIAM Journal on Computing, 8(2):175–198, 1979.
- [42] D. Merkl, S. Hui-He, M. Dittenbach, and A. Rauber. Adaptive hierarchical incremental grid growing: An architecture for high-dimensional data visualization. In In Proceedings of the 4th Workshop on Self-Organizing Maps, Advances in Self-Organizing Maps, pages 293–298, 2003.

- [43] R. Miikkulainen. Script recognition with hierarchical feature maps. Connection Science, 2(1&2):83–101, 1990.
- [44] R. L. Ogniewicz and O. Kübler. Hierarchic voronoi skeletons. Pattern Recognition, 28(3):343–359, 1995.
- [45] M. Oja, S. Kaski, and T. Kohonen. Bibliography of self-organizing map (SOM) papers: 1998-2001 addendum. Neural Computing Surveys, 3:1–156, 2003.
- [46] J. Pakkanen, J. Iivarinen, and E. Oja. The Evolving Tree a novel self-organizing network for data analysis. *Neural Processing Letters*, 20(3):199–211, December 2004.
- [47] G. Peano. Sur une courbe, qui remplit toute une aire plane. Mathematische Annalen, 36(1):157–160, 1890.
- [48] M. Pöllä, T. Honkela, and T. Kohonen. Bibliography of self-organizing map (SOM) papers: 2002-2005 addendum. Technical Report TKK-ICS-R23, Helsinki University of Technology, Department of Information and Computer Science, Espoo, Finland, December 2009.
- [49] G. Pölzlbauer. Survey and comparison of quality measures for self-organizing maps. In Ján Paralič, Georg Pölzlbauer, and Andreas Rauber, editors, Proceedings of the Fifth Workshop on Data Analysis (WDA'04), pages 67–82, Sliezsky dom, Vysoké Tatry, Slovakia, June 24–27 2004. Elfa Academic Press.
- [50] A. Rauber, D. Merkl, and M. Dittenbach. The Growing Hierarchical Self-Organizing Map: exploratory analysis of high-dimensional data. *IEEE Transactions on Neural Networks*, 13(6):1331–1341, 2002.
- [51] R. Rojas. Neural networks: a systematic introduction. Springer-Verlag New York, Inc., New York, NY, USA, 1996.
- [52] E. V. Samsonova, J. N. Kok, and A. P. IJzerman. Treesom: Cluster analysis in the self-organizing map. *Neural Networks*, 19(6–7):935 – 949, 2006. Advances in Self Organising Maps - WSOM'05.
- [53] R. Singh, V. Cherkassky, and N. Papanikolopoulos. Self-Organizing Maps for the skeletonization of sparse shapes. *Neural Networks, IEEE Transactions on*, 11(1):241–248, Jan 2000.
- [54] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. J. ACM, 32(3):652–686, 1985.
- [55] J. Venna and S. Kaski. Neighborhood preservation in nonlinear projection methods: An experimental study. In Georg Dorffner, Horst Bischof, and Kurt Hornik, editors, *ICANN*, volume 2130 of *Lecture Notes in Computer Science*, pages 485–491. Springer, 2001.
- [56] K. C. Yao, M. Mignotte, C. Collet, P. Galerne, and G. Burel. Unsupervised segmentation using a selforganizing map and a noise model estimation in sonar imagery. *Pattern Recognition*, 33(9):1575 – 1584, 2000.