

# On Using Adaptive Binary Search Trees to Enhance Self Organizing Maps

César A. Astudillo<sup>1</sup> and B. John Oommen<sup>2</sup>

<sup>1</sup> Universidad de Talca, Curicó, Chile  
castudillo@utalca.cl

<sup>2</sup> Carleton University, Ottawa, Canada  
oommen@scs.carleton.ca

**Abstract.** We present a strategy by which a Self-Organizing Map (SOM) with an underlying Binary Search Tree (BST) structure can be adaptively re-structured using conditional rotations. These rotations on the nodes of the tree are *local* and are performed in *constant time*, guaranteeing a decrease in the Weighted Path Length (WPL) of the entire tree. As a result, the algorithm, referred to as the Tree-based Topology-Oriented SOM with Conditional Rotations (TTO-CONROT), 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.

## 1 Introduction

Even though numerous researchers have focused on deriving variants of the original Self-Organizing Map (SOM) strategy, few of the reported results possess the ability of modifying the underlying topology, leading to a dynamic modification of the structure of the network by adding and/or deleting nodes and their inter-connections. Moreover, only a small set of strategies use a tree as their underlying data structure [1,2,3,4]. From our perspective, we believe that it is also possible to gain a better understanding of the unknown data distribution by performing *structural* tree-based modifications on the tree, by rotating the nodes within the Binary Search Tree (BST) that holds the whole structure of neurons. Thus, we attempt to use rotations, tree-based neighbors *and* the feature space as an effort to enhance the capabilities of the SOM by representing the underlying data distribution and its structure more accurately. Furthermore, as a long term ambition, this might be useful for the design of faster methods for locating the Best Matching Unit (BMU).

One of the primary goals of Adaptive Data Structures (ADS) is to seek an optimal arrangement of the elements, by automatically reorganizing the structure itself so as to reduce the average access time. The solution to obtain the *optimal* BST is well known when the access probabilities of the nodes are known beforehand [5]. However, our research concentrates on the case when these access probabilities are *not known a priori*. In this setting, the most effective solution

is due to Cheetham *et al.* and uses the concept of conditional rotations [6]. The latter paper proposed a philosophy where an accessed element is rotated towards the root if and only if the overall Weighted Path Length (WPL) of the resulting BST decreases.

The strategy that we are presently proposing, namely the Tree-based Topology-Oriented SOM with Conditional Rotations (TTO-CONROT), has a set of neurons, which, like all SOM-based methods, represents the data space in a condensed manner. Secondly, it possesses a connection between the neurons, where the neighbors are based on a learned nearness measure that is tree-based. Similar to the reported families of SOMs, a subset of neurons closest to the BMU are moved towards the sample point using a vector-quantization (VQ) rule. However, unlike most of 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 TTO-CONROT incorporates tree-based mutating operations, namely the above-mentioned conditional rotations.

Our proposed strategy is adaptive, with regard to the migration of the points *and* with regard to the identity of the neurons moved. Additionally, the distribution of the neurons in the feature space mimics the distribution of the sample points. Lastly, by virtue of the conditional rotations, it turns out that the entire tree of neurons 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.

The contributions of the paper can be summarized as follows. First, we present an integration of the fields of SOMs and ADS. Secondly, the neurons of the SOM are linked together using an underlying tree-based data structure, and they are governed by the laws of the Tree-based Topology-Oriented SOM (TTOSOM) paradigm, and simultaneously by the restructuring adaptation provided by conditional rotations (CONROT). Third, the adaptive nature of TTO-CONROT 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. Finally, we explain how the set of neurons in the proximity of the BMU is affected as a result of applying the rotations on the BST.

The rest of the paper is organized as follows. The next section surveys the relevant literature, 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 3, we provide an in-depth explanation of the TTO-CONROT 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.

## 2 Literature Review

A number of variants of the original SOM [7] have been presented through the years, attempting 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 search of the BMU. We focus our attention on a specific family of enhancements in which the neurons are inter-connected using a tree topology [1,2,3,4]. In [1] the authors presented a tree-based SOM called the Growing Hierarchical SOM (GHSOM), in which each node corresponds to an independent SOM, and where dynamic behavior is manifested by adding rows or columns to each SOM depending on a suitable criterion. The authors of [2] have studied a variant of the SOM called the Self-Organizing Tree Map (SOTM), which also utilizes a tree-based arrangement of the neurons, and which uses the distance in the *feature* space to determine the BMU. In [3] the authors proposed a tree-structured neural network called the evolving tree (ET), which takes advantage of a sub-optimal procedure to determine 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, which is a non-leaf node that does not participate in the training process, and which is thus removed from the Bubble of Activity (BoA).

Here, we focus on the TTOSOM [4]. The TTOSOM incorporates the SOM with a tree which has an arbitrary number of children. Furthermore, it is assumed that the user has the ability to describe such a tree, reflecting the *a priori* knowledge about the *structure* of the data distribution<sup>1</sup>. The TTOSOM also possesses a BoA with particular properties, considering the distance in the tree space, where leaves and non-leaf nodes are part of this neighborhood. Another interesting property displayed by the TTOSOM is its ability to reproduce the results obtained by Kohonen [7], 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 did [4]. Additionally, if the original topology of the tree followed the overall shape of the data distribution, the results reported in [4] showed that it is also possible to obtain a symmetric topology for the codebook vectors.

We shall now proceed to describe the corresponding relevant work in the field of the tree-based ADS. A BST may be used to store records whose keys are members of an ordered set. In this paper, we are in the domain where 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 [8]. A few memory-less tree reorganizing schemes<sup>2</sup> which use this operation have been presented in the literature. In the Move-to-Root Heuristic [12], each time a record is accessed, rotations are performed on it in an upwards direction until it becomes the root of the tree. On the other hand, the simple Exchange rule [12] rotates the accessed element one level towards the root. Sleator and Tarjan [13] introduced a technique, which also moves the accessed record up to the root of the tree using a restructuring operation. Their

---

<sup>1</sup> The beauty of such an arrangement is that the data can be represented in multiple ways depending on the specific perspective of the user.

<sup>2</sup> This review is necessary brief. A more detailed version is found in [9,10,11].

structure, called the Splay Tree, was shown to have an amortized time complexity of  $O(\log N)$  for a complete set of tree operations. 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 [14] and Mehlhorn's D-Tree [15]. In spite of all their advantages, all of the schemes mentioned above have drawbacks, some of which are more serious than others.

This paper focuses on a particular heuristic, namely, the Conditional Rotations (CONROT-BST) [6], which has been shown to reorganize a BST so as to asymptotically arrive at an optimal form. CONROT-BST 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 1, formally given below, describes the process of the Conditional Rotation 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 (assumed to be present in the tree).

---

**Algorithm 1.** 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
3:   if is-left-child( $j$ ) = TRUE then
4:      $\Psi_j \leftarrow 2\tau_j - \tau_{jR} - \tau_{P(j)}$ 
5:   else
6:      $\Psi_j \leftarrow 2\tau_j - \tau_{jL} - \tau_{P(j)}$ 
7:   end if
8:   if  $\Psi_j > 0$  then
9:     rotate-upwards( $j$ )
10:    recalculate-tau( $j$ )
11:    recalculate-tau( $P(j)$ )
12:   end if
13:   return record  $j$ 
14: else
15:   if  $k_i < k_j$  then
16:     CONROT-BST( left-child( $j$ ) ,  $k_i$  )
17:   else
18:     CONROT-BST( right-child( $j$ ) ,  $k_i$  )
19:   end if
20: end if

```

**End Algorithm**

---

We define  $\tau_i(n)$  as the total number of accesses to the subtree rooted at node  $i$ . CONROT-BST computes the following equation to determine the value of a quantity referred to as  $\Psi_j$ , for a particular node  $j$ , where:

$$\Psi_j = \begin{cases} 2\tau_j - \tau_{jR} - \tau_{P(j)} & \text{if } j \text{ is a left child of } P(j) \\ 2\tau_j - \tau_{jL} - \tau_{P(j)} & \text{if } j \text{ is a right child of } P(j) \end{cases} \quad (1)$$

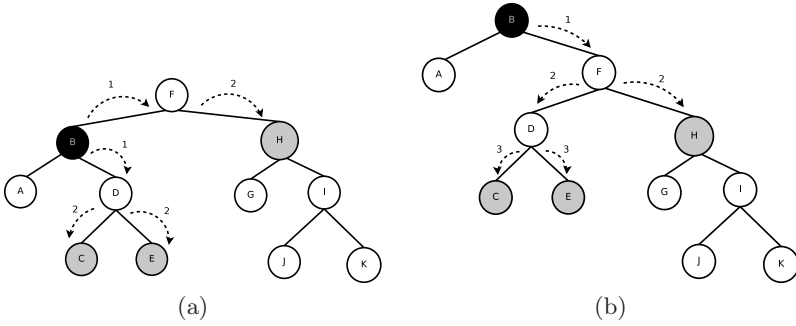
When  $\Psi_j$  is less than zero, an upward rotation is performed. The authors of [6] have shown that this single rotation yields to a decrease in the WPL of the *entire* tree. Once the rotation takes place, it is necessary to update the corresponding counters,  $\tau$ . Fortunately this task only involve the updating of  $\tau_i$ , for the rotated node, and the counter of its parent,  $\tau_{P(i)}$ . 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 Merging ADS and TTOSOM

This section concentrates on the details of the integration between the fields of ADS and the SOM. More specifically we shall concentrate on the integration of the CONROT-BST heuristic [6] into a TTOSOM [4], both of which were explained in the previous section. 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**.

As in the case of the TTOSOM [4], the *Neural Distance*,  $d_N$ , between two neurons is defined as the minimum number of edges required to go from one to the other. 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 TTO-CONROT employs a tree which is dynamically modified, where 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. This renders the resultant SOM to be unique and distinct from the state-of-the-art.

Fig. 1 presents 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  $B$ . Fig. 1a illustrates the situation prior to an access, where 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. Secondly, Fig. 1b 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.



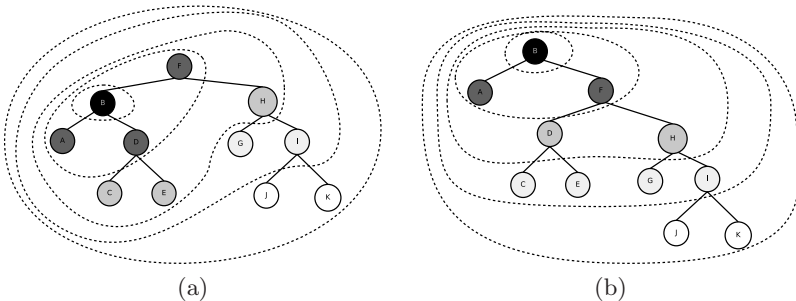
**Fig. 1.** Example of the Neural Distance before and after a rotation

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. The BoA can be formally defined as [4]

$$B(v_i; T, r) = \{v \mid d_N(v_i, v; T) \leq r\}, \tag{2}$$

where  $v_i$  is the node currently being examined, and  $v$  is an arbitrary node in the tree  $T$ , whose nodes are  $V$ .

Fig. 2 depicts how the BoA differs from the one defined by the TTOSOM as a result of applying a rotation. Fig. 2a shows the BoA around the node  $B$ , using the same configuration of the tree as in Fig. 1a, i.e., before the rotation takes place. Here, the BoA when  $r = 1$  involves 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  $r = 4$  case leads to a BoA which includes the whole set of nodes. Now, observe the case presented in Fig. 2b, which corresponds to the BoA around  $B$  after the rotation upwards has been effected, i.e. the same configuration of the tree used in Fig. 1b. 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



**Fig. 2.** Example of BoA before and after a rotation is invoked at node  $B$

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  $r = 4$ , the BoA invokes the entire tree.

The CONROT-BST heuristic [6] requires that the tree should possess the BST property:

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

To satisfy the BST property, first of all we see that, the tree must be binary<sup>3</sup>. The tree trained by the TTOSOM is restricted to contain at most two children per node and a comparison operator between the two children is considered. 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 lexicographical arrangement of the nodes.

It happens that the concept of the “just accessed” node in the CONROT-BST is compatible with the corresponding BMU defined for the Competitive Learning (CL) model. 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. 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 position<sup>4</sup> in the network with respect to the other neurons in the neural network. Thus, while “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.

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 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 inclusion of the rotations renders this dynamic.

In our proposed approach, the codebooks of the SOM correspond to the nodes of a BST. Apart from the information regarding the codebooks themselves, each neuron requires the maintenance of additional fields to achieve the adaptation.

---

<sup>3</sup> Of 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.

<sup>4</sup> As 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.

Also, besides the codebook vectors, 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.

The training module of the TTO-CONROT is responsible for determining the BMU, performing restructuring, calculating the BoA and migrating the neurons within the BoA. Basically, what it achieves, is to integrate the CONROT algorithm in the sequence of steps of the Training phase defined by the TTOSOM. Algorithm 2 describes the details of how this integration is fulfilled. Algorithm 2 receives as input a sample point,  $x$ , and the pointer to the root of the tree,  $p$ . Line No. 1, performs the first task of the algorithm, which involves the determination of the BMU. After that, line No. 2, deals with the call to the CONROT algorithm. The reason why we follow this sequence of steps is that the parameters needed to perform the conditional rotation, as specified in [6], includes the key of the element queried, which, in the present context, corresponds to the key of the BMU. At this stage of the algorithm, the BMU may be rotated or not, and the BoA is determined *after* this restructuring process, which is performed in lines No. 3 and 4 of the algorithm. Finally, lines No. 5 to 7, are responsible for the neural migration, involving the movement of the neurons within the BoA towards the input sample.

---

**Algorithm 2.** TTO-CONROT-BST\_train( $x,p$ )

---

**Input:**

- i)  $x$ , a sample signal.
- ii)  $p$ , the pointer to the tree.

**Method:**

- 1:  $v \leftarrow \text{TTO-SOM\_Find\_BMU}(p,x,p)$
- 2:  $\text{cond-rot-bst}(p,v.\text{getID}())$
- 3:  $B \leftarrow \{v\}$
- 4:  $\text{TTO-SOM\_Calculate\_Neighborhood}(B,v,\text{radius})$
- 5: **for all**  $b \in B$  **do**
- 6:      $\text{update\_rule}(b.\text{getCodebook}(),x)$
- 7: **end for**

**End Algorithm**

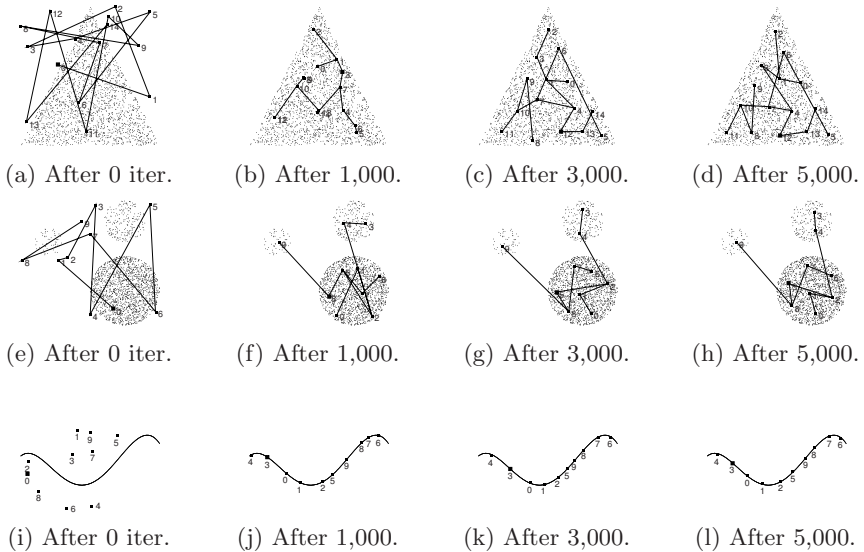
---

Even though, we have used the advantages of the CONROT algorithm, the architecture that we are proposing allows us to utilize an alternative restructuring module. Candidates which can be used to perform the adaptation are the ones mentioned in Section 2, and include the splay and the monotonic-tree algorithms, among others [11].

## 4 Experimental Results

To illustrate the capabilities of our method, the experiments reported in the present work are limited to the two-dimensional feature space. However, it is important to remark that the algorithm is also capable of solving problems in higher dimensions, though a graphical representation of the results is not as illustrative. As per the results obtained in [4], the TTOSOM is capable of inferring the distribution and structure of the data. In our present work, we are interested in knowing the effects of applying the neural rotation as part of the training process. The experiments briefly presented in this section use the same schedule for the learning rate and radius, i.e., no particular refinement of the parameters has been done to each particular data set.

First, we consider the data generated from a triangular-shaped distribution, as shown in Figs. 3a-3d. In this case, the initial tree topology is unidirectional. For the initialization phase a 1-ary tree (i.e., a list) is employed as the special case of the structure, and the respective keys are assigned in an increasing order. At the beginning, the prototype vectors are 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 and reallocated (see Figs. 3b and 3c), the 1-ary tree is modified as a consequence of the rotations. Finally, Fig. 3d depicts the case after convergence has taken place. Here, the tree nodes are uniformly distributed over the whole triangular shape. The BST property is still preserved, and further rotations are still possible. This experiment serves as an excellent example to show the differences with respect to the original TTOSOM algorithm



**Fig. 3.** A 1-ary tree, i.e. a list topology, learns different distributions using the TTOCONROT algorithm after utilizing the *same* set of parameters

[4], where a similar data set was utilized. In the case of the TTO-CONROT the points effectively represent the whole data set. Here, no particular *a priori* information about the structure of the data distribution is necessary; rather, this is learnt during the training process, as shown in Fig. 3d. In this manner, the specification of the initial tree topology required by the TTOSOM is no longer needed, and an alternative specification, which only requires the number of nodes in the initial 1-ary tree, is sufficient.

Another experiment is the one shown in Figs. 3e-3h, which entails a data set generated from 3 circular-shaped clouds where the circles possess a different size and density. In this experiment, again, in the first iteration, the original structure of 1-ary tree is lost because of the random selection of the codebook vectors. Interestingly, after convergence, and as depicted in Fig. 3h, the algorithm places a proportional number of codebook vectors in each of the three circles according to the density of their data points.

Lastly, Figs. 3i-3l demonstrate the power of the scheme for a linear curve.

## 5 Conclusions

In this paper, we have proposed a novel integration between the areas of ADS and the SOM. In particular, we have shown how a tree-based SOM can be adaptively transformed by the employment of an underlying BST structure and subsequently, re-structured using rotations that are performed conditionally. Our proposed method is able to infer the topological properties of the stochastic distribution, and at the same time, attempts to build the best BST that represents the data set.

## References

1. Rauber, A., Merkl, D., Dittenbach, M.: The Growing Hierarchical Self-Organizing Map: exploratory analysis of high-dimensional data. *IEEE Transactions on Neural Networks* 13(6), 1331–1341 (2002)
2. Guan, L.: Self-organizing trees and forests: A powerful tool in pattern clustering and recognition. In: Campilho, A., Kamel, M.S. (eds.) *ICIAR 2006, Part I. LNCS*, vol. 4141, pp. 1–14. Springer, Heidelberg (2006)
3. Pakkanen, J., Iivarinen, J., Oja, E.: The Evolving Tree — a novel self-organizing network for data analysis. *Neural Processing Letters* 20(3), 199–211 (2004)
4. Astudillo, C.A., Oommen, B.J.: A novel self organizing map which utilizes imposed tree-based topologies. In: *6th International Conference on Computer Recognition Systems*, vol. 57, pp. 169–178 (2009)
5. Knuth, D.E.: *The art of computer programming*, 2nd edn. Sorting and searching, vol. 3. Addison Wesley Longman Publishing Co., Inc., Redwood City (1998)
6. Cheetham, R.P., Oommen, B.J., Ng, D.T.H.: Adaptive structuring of binary search trees using conditional rotations. *IEEE Trans. on Knowl. and Data Eng.* 5(4), 695–704 (1993)
7. Kohonen, T.: *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus (2001)

8. Adelson-Velskii, M., Landis, M.E.: An algorithm for the organization of information. *Sov. Math. DokL* 3, 1259–1262 (1962)
9. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: *Introduction to Algorithms*, 2nd edn. McGraw-Hill Science/Engineering/Math., New York (2001)
10. Lai, T.W.H.: *Efficient maintenance of binary search trees*. PhD thesis, University of Waterloo, Waterloo, Ont., Canada (1990)
11. Astudillo, C.A., Oommen, B.J.: Self organizing maps whose topologies can be learnt with adaptive binary search trees using conditional rotations. Journal version of this paper (2009) (submitted for publication)
12. Allen, B., Munro, I.: Self-organizing binary search trees. *J. ACM* 25(4), 526–535 (1978)
13. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. *J. ACM* 32(3), 652–686 (1985)
14. Bitner, J.R.: Heuristics that dynamically organize data structures. *SIAM J. Comput.* 8, 82–110 (1979)
15. Mehlhorn, K.: Dynamic binary search. *SIAM Journal on Computing* 8(2), 175–198 (1979)