Speeding up algorithms of SOM Family for Large and High Dimensional Databases

Ernesto Cuadros-Vargas[1], Roseli Ap. Francelin Romero[1], Klaus Obermayer[2]
[1]ICMC-University of São Paulo, Cx Postal 668 São-Carlos-SP, Brazil
[2]Dept. of Electrical Engineering and Computer Science
Technische Universität Berlin, 10587 Berlin, Germany
+55-16-273-9661, FAX +55-16-273-8118

{ecuadros, rafrance}@icmc.usp.br, oby@cs.tu-berlin.de
Keywords: Spatial Access Methods, Self-Organizing Maps

Abstract— In this paper, Spatial Access Methods, like R-Tree and k-d Tree, for indexing data, are used to speed up the training process and performance of data analysis methods which learning algorithms are kind of competitive learning. Often, the search for the winning neuron is performed sequentially, which leads to a large number of operations. Instead of using the common sequential determination of the winning neuron, which has a computational complexity of $O(N)$ (where N is the number of candidate units to be the winner), the approach proposed here allows to find the winning neuron in, approximately, $\log N$ steps. Results obtained by incorporating k-d-tree, R-Tree into Self-Organizing Maps are presented and compared with their sequential counterpart implementation of SOM. The methods of SOM family used are: k-means, Kohonen network and GNG network. Several database has been used for demonstrating that a dramatic speed up can be achieved, what is very significant when large-scale and high dimensional databases are being considered.

1 Introduction

Many artificial neural network models adopt the competitive learning for updating their weights. This kind of learning algorithm finds to the winning among several units, in a sequential way, which leads to a large number of operations.

Self-Organizing Maps (SOMs) are example of such networks. They are methods for data analysis which combine the grouping of data with an embedding in a low dimensional space for the purpose of visualization. SOMs include standard clustering algorithms (k-Means) as the limiting case that - for every data point only one - weight vector is updated at one iteration. Let us now consider the case of metric data, i.e. the case that every data point is characterized by a feature vector. Assignment of data points to clusters is performed by (i) calculating the distance between the data point and the weight vector of the units, and (ii) by assigning the data point to the unit whose weight vector is closest. In general, distance calculations have to be performed for every data point and for every iteration during the learning process, and can be computationally expensive in particular if the dimension of data space is high. The involved computational cost has a computational complexity of $O(N)$ and this can imply to a high computation time when these methods are applied to large-scale databases.

On the other hand, Spatial Access Methods (SAM) [11], which are used for information retrieval in large databases [1], perform a hierarchical partitioning of input space and arrange the partitions in a tree structure. Then this tree can also be used as a search tree for to retrieve objects, like the weight vector of a unit in SOM. If the tree is constructed in an efficient way, the number of visited nodes could be $O(\log_m(N))$ where N is the number of vectors inserted in the neural network and m is the number of entries per node.

In [7], it was introduced a family of dual $k$dtree traversal algorithms for accelerating a wide class of statistical methods that are naively quadratic in the number of datapoints. However, it was used only the k-d-tree structure [2], which is widely known as an inefficient SAM due to the unbalanced tree generated and it was not address potential speedups during learning and adaptation.

It is known that one standard method of spatial indexing is the R-Tree [8], which is frequently considered as the main reference method for new SAMs. R-Trees provide good results if the dimension of the feature space is lower than approximately 20. If the dimension is larger than this, other techniques like OMNI [12] should be preferred. The reader can see [11, 3] for more references about these and other access methods.

Recently, a SAM-SOM family has been proposed by us [4] in which R-Tree has been used to reduce dramatically the number of comparisons of each of the N points in a dataset with each other point. It has been proposed an improved search procedure applied to SOM, based on R-Trees, which reduces the computational complexity for $log(N)$ instead of $O(N)$ in searching for the winning neuron.

In the present paper, this idea has been extended

and a comparative analysis about the performance of SOM (using R-Tree and k-d Tree for choosing the winning neuron) and the traditional SOM implementation is presented using several databases. Three algorithms of SOM family, including: K-Means Clustering, Kohonen Map and Growing Neural Gas are addressed in this paper. Further, it will be shown how to incorporate SAM into SOM for both batch and on-line learning methods aiming to speed up the training process for neural networks with competitive learning. It is worth to note that SOMs serve only as a benchmark example. The approach proposed can be applied to a variety of algorithms which involve distance computations among data objects.

This paper is organized as follows. In section 2, it will be briefly summarized the previous work. In the section 3, it will be shown how the incorporation of SAM to SOM is realized for both batch-learning and on-line learning. The benchmark results are presented in section 4. Finally, in section 5, conclusions and future work are presented.

2 Related works

Let us consider a data set, where the data objects i are described by feature vectors $\vec{x}_i \in \mathbb{R}^n$. The data objects should be represented by a SOM network with M units (or neurons) p, with weight vectors $\vec{w}_p$. Given a set of training data, the weight vectors can be determined using either batch [9] or on-line learning [10]. The batch learning algorithm for the standard SOM involves an iteration of the following two steps in its inner loop:

$$p(\vec{x_k}) = \text{argmin}_q|\vec{x}_k - \vec{w}_q|, \forall k, \quad \text{assignment step} \quad (1)$$

$$\Delta\vec{w}_q = \epsilon h_{qp(\vec{x_k})}(\vec{x}_k - \vec{w}_q), \forall k, q \quad \text{adaptation step} \quad (2)$$

where ϵ is the learning constant and h_{qp} is the neighborhood function. In on-line learning, one data point is processed at a time, and we obtain:

$$p(\vec{x_k}) = \text{argmin}_q|\vec{x}_k - \vec{w}_q|, \quad \text{assignment step} \quad (3)$$

$$\Delta\vec{w}_q = \epsilon h_{qp(\vec{x_k})}(\vec{x}_k - \vec{w}_q), \quad \forall q \text{ adaptation step} \quad (4)$$

If $h_{qp} = \delta_{qp}$, i.e. if only one unit is being updated for each data point, eqs. (1)-(4) reduce to the batch and the on-line version of the k-Means clustering algorithm [9, 10]. If a new data point $\vec{x}_l$ is presented after learning, the data point is assigned to the unit $p(\vec{x}_l)$ with the weight vector $\vec{w}_p$ which is closest, i.e.

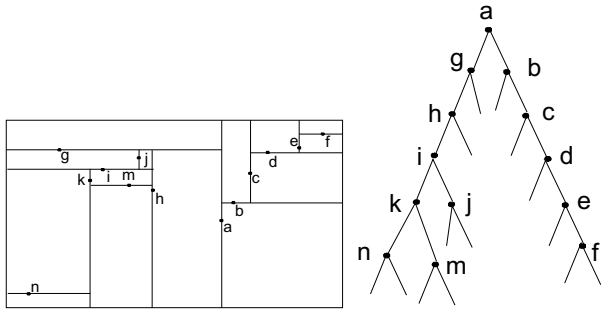
$$p(\vec{x_k}) \quad = \quad \text{argmin}_q|\vec{x}_l - \vec{w}_q|. \quad (5)$$

For simplicity, the Euclidean distances has been adopted, but all results presented in section 4 also hold if other distance measures are used as well as if eqs. (1)-(4) are changed accordingly to different neighborhoods.

The Growing Neural Gas (GNG) [6] is a variant of SOM, for which nodes are added during the learning process and for which the neighborhood function is adaptive. It is an incremental clustering method. Because it has been also used for testing the proposed approach.

On the other hand, Spatial Access Methods-SAM are methods of indexing data objects, which are described by feature vectors like $(x_1, x_2, ..., x_n)$. There are many methods to index multi-dimensional objects. Some of the most classics SAM are k-d Tree [2] and R-Tree [8] which are briefly explained to follow.

- **k-d Tree** [2]: this SAM is used for n-dimensional vectors $(x_1, x_2, ..., x_n)$. It creates a binary tree using the first co-ordinate, x_1, to decide its position (left or right) at the first level, the co-ordinate x_2 will be used for the 2nd level. When all the co-ordinates have been used, x_1 is used again and so on. In the Figures 1(a) and 1(b), it is shown an example of this structure for 2 dimensional data (2-d Tree).



(a) Spatial representation (b) Internal representation

Figure 1: 2-d Tree.

- **R-Tree** [8]: This is the first non point-based SAM. It could be seen as a generalization of B-Trees for multidimensional data indexing. In this tree the information is in the leaves. Each upper level has the information about the Minimum Bounding Rectangle (MBR) necessary to contains all the child nodes.
  Figures 2 and 3 illustrate, how R-Trees organize the spatial data. Each node, except the leaves, contains information about the Minimum Bounding Rectangle (MBR) that contains all its child nodes. In Figure 2, for example, we can see that $R17$, $R18$ and $R19$ are contained in $R7$. Consequently, those three objects are represented as child nodes of the parent node $R7$ in the tree (Figure 3). If we insert a new element in this tree (Figure 3), the search algorithm begins on the top of the tree and inspects either $R1$ or $R2$ according with the intersection between the new object and the existing rectangles. If the new object is not contained by one existing rectangles, the solution

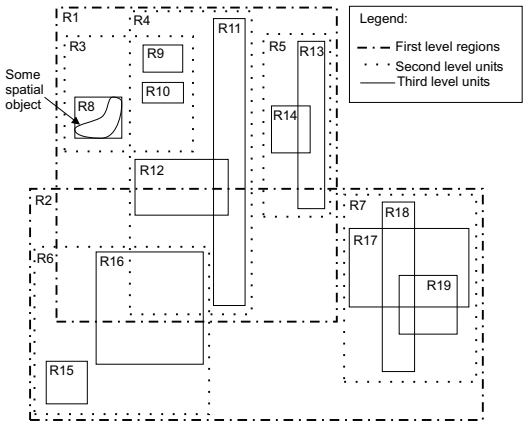


Figure 2: Minimum bounding regions represented by a R-Tree

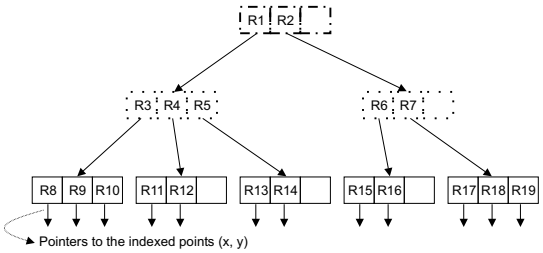


Figure 3: R-Tree Internal representation.

is to increase the borders of one of them. The rectangle which is selected to expand its borders is the one for which the added area is minimal. As it can be seen in Fig. 3, each page can contain a maximum number of nodes (three in this case) which is an initial parameter to be fixed before the tree is build. If we insert a new object which should go under $R7$ the result is an overflow. The solution is to split this node in two new nodes containing 2 objects each one. The MBR represented by $R7$ is removed and two new MBR are inserted instead.

3 Incorporating SAM into SOM

In order to incorporate k-d Tree or R-Tree or any other access methods to SOMs, it is necessary to identify each leaf of the tree with one unit (cluster) of the competitive network. In [4], an algorithm has been proposed for incorporating R-Tree into Growing Neural Gas. This algorithm establishes a correspondence between each neuron in GNG with a leaf in R-Tree. The nearest neighbor is determined by searching into R-Tree. As explained in [8], the height of a R-Tree with m entries per node and N vectors inserted is, at most, $\lceil log_m(N) \rceil - 1$. For using k-d Tree or any other SAM, the principle is the same, that is, the nearest neighbor is determined by searching into the corresponding SAM. The only difference will be the computational cost related to internal specific algorithm to retrieve the information.

During the learning process, however, weight vectors are constantly modified and it is necessary to rearrange objects into the data structure. In the case of k-d Trees, the main drawback is that, this access method does not generate a balanced tree which increases considerably the time for information retrieval.

The incorporation of SAM into SOM can be realized in the following way. For batch-learning, where all weight vectors are changed simultaneously, the SAM being used should be built after every iteration. For on-line learning, where only a few weight vectors are changed at a time, the corresponding leaves of the SAM should be updated.

In order to find the nearest neighbor for a new pattern presented to neural network, it can be used either a conventional algorithm for k Nearest Neighbor [13] or a Range Query with an initial radius and increase it gradually based on any previous knowledge about the dataset. The latter technique is that one used in all our experiments.

4 Experimental Results

In this section, the performance of the proposed technique is verified through several experiments realized. The results has been obtained through the following techniques: k-Means clustering (batch and on-line learning), standard Kohonen network, and the GNG network. For the two latest techniques we have used on-line learning.

All algorithms, including k-d Tree and R-Tree , were implemented using Microsoft Visual C++ 6.0 on a PC 1 GHz, 256 Mb RAM, running Windows XP. The number of distance calculations was used as a performance measure to compare R-Tree indexing with the sequential search for the winning unit.

The methods were applied to the four datasets:

Abalone: This dataset is composed by 4177 vectors 8-d extracted from abalon's physical measuremente. The file was obtained from `UCI-Irvine repository of machine learning databases and domain theories`[1]

Letters: This dataset is composed by 20986 17-d vectors which describe the shape of handwritten letters. The file was obtained from `UCI-Irvine repository of machine learning databases and domain theories`[2]

Faces: This dataset is composed by 11900 16-d feature vectors which describe the properties of faces. This dataset was obtained from from the Informedia Project at Carnegie Mellon University.

[1] `ftp://ftp.ics.uci.edu/pub/machine-learning-databases/abalone/`.

[2] `ftp://ftp.ics.uci.edu/pub/machine-learning-databases/letter-recognition/`.

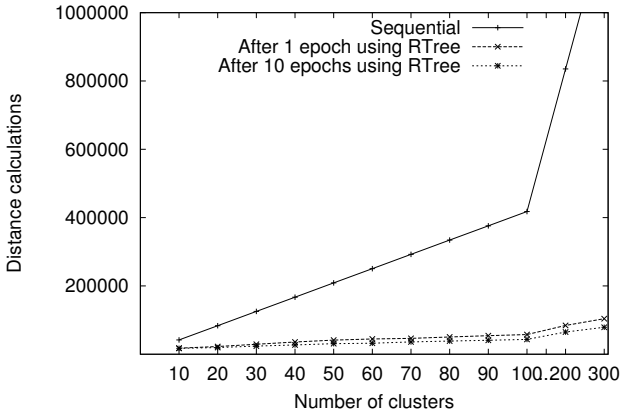
Images: This dataset is composed by 20000 1215-dimensional feature vectors which describe the properties of images. The data was preprocessed using the FastMap algorithm [5] in order to obtain 20-dimensional feature vectors. This dataset was obtained from from the Informedia Project at Carnegie Mellon University.

In all tests realized, it has not been used k-d Trees, for online algorithms, because this structure has serious restrictions to move and remove elements. When an online algorithm is being executed, it is necessary to move elements all the time, and when an element, which is not a leaf, is moved using k-d Trees, it could jump from a branch to another. When this happens, the element should be removed and inserted again in its new branch. The problem arises by the removal process because it can force to rebuild the whole tree. This situation is quite common during online training processes. This is the main reason by which k-d Trees has been used only for batch processes. More information about this problem could be found in [2, 11].

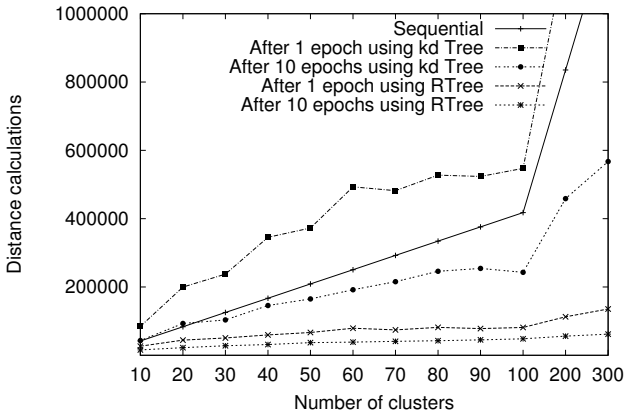
In Fig. 4, 5 and 6 are presented the results obtained by using k-means in Abalone, Faces and Letters datasets, respectively.

The x-axis represents the number of prototypes used during the simulation and y-axis represents the number of distance calculations required to find the nearest neighbor unit using k-Means algorithm. As it can be seen in these figures, using the on-line version, the number of distance calculations required is always less than using the batch considering the same dataset. It is due to the fact that, in on-line version, the weights are updated after each pattern.

In Figure 6(b), it is noted a better performance for sequential approach during the first epoch. This result is probably due to the proposed algorithm to find the k-nearest neighbors be based on range queries with an estimated initial radius. At the beginning of the process, all the units are chosen randomly and it is necessary to expand the initial radius more than the normal. However, only one epoch is necessary in order to improve this result. The same problem does not appear with on-line k-Means algorithm. The probably reason is that the prototypes are constantly modified for each pattern. As it can be seen in the previous figures, there is a big difference between the sequential approach and the k-Means algorithm using R-Tree. R-Tree has a better performance than k-d tree because it creates a tree balanced. It does not happen for k-d Tree algorithm which can create a degenerated binary tree similar to a linked list. This unbalanced tree has consequences over information retrieval because it will be necessary to inspect several branches. Even considering drawbacks presented by k-d trees, both data structures, k-d trees and R-Tree, present better performance than sequential implementation considering batch learning.



(a) on-line



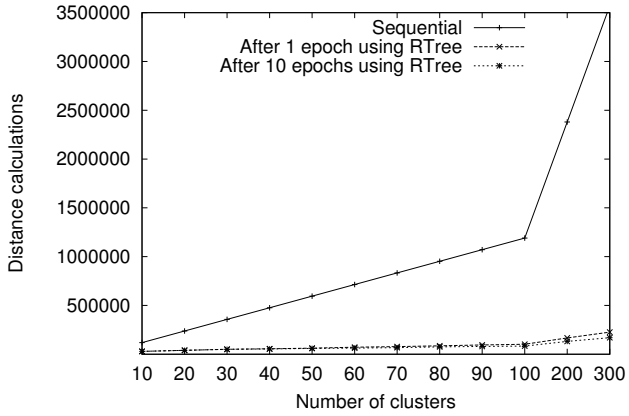
(b) batch

Figure 4: Number of distance calculations as functions of the number of prototypes using k-Means with the Abalone dataset.

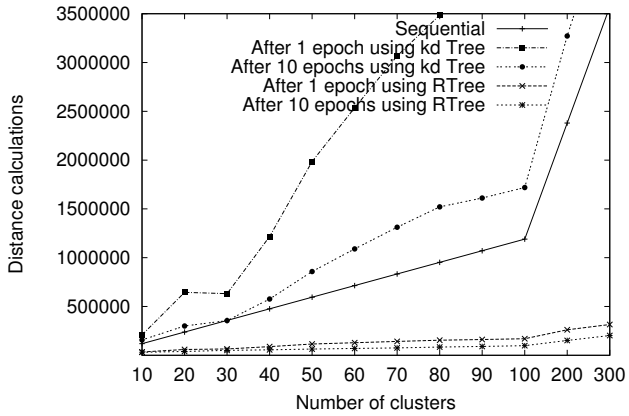
For all these figures, it is important to note that, after 90 and 100, in the x-axis, the next represented points correspond to 200 and 300 clusters.

In Figure 7 are presented results obtained considering a Kohonen network. It is shown the number of distance calculations required during the training of a SOM for both the sequential implementation and the implementation SOM-RTree (that is using R-Tree). It is important to pay attention to the following fact: the SOM network size used in this case is small (only 10x10 units) but the difference in terms of performance between the SOM sequential implementation and the SOM-RTree implementation is big. For large databases the number of units necessary will be more much more than 10x10 units.

In contrast to the results presented by Kohonen network, the results using GNG (Figure 8) are presented by a curve because the number unit is not constant, since this neural network is a constructive network. As it can be seen, in Figure 8(a) both results are the same until GNG gets an architecture constituted by 10

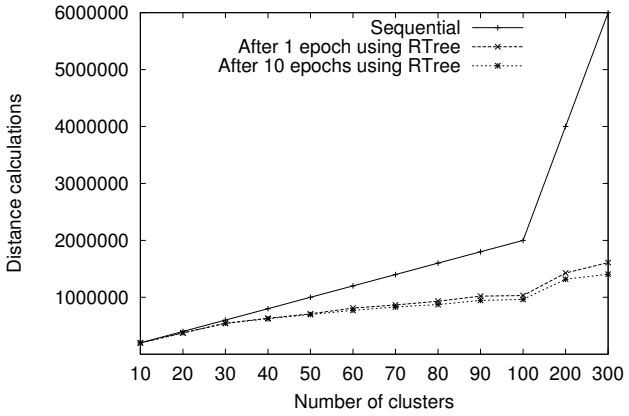


(a) on-line

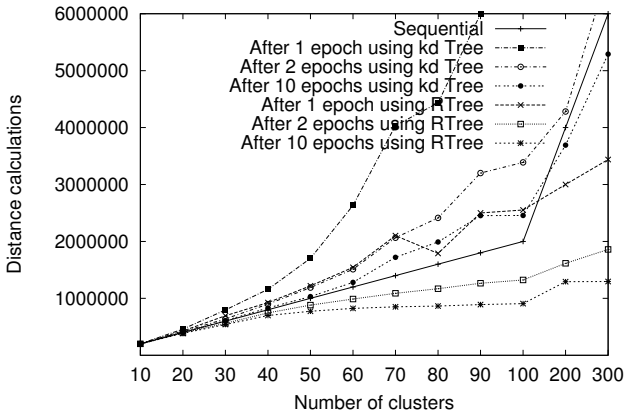


(b) batch

Figure 5: Number of distance calculations as function of the number of prototypes using k-Means with the Faces dataset.



(a) on-line



(b) batch

Figure 6: Number of distance calculations as function of the number of prototypes using k-Means with Letters dataset.

units. It is because the GNG is not using the R-Tree before this point but sequential approach (the number 10 was chosen considering that, when the R-Tree is too small, it will be necessary to inspect all the pages, probably less than three at that instant, and it is more expensive to use the R-Tree instead of the sequential approach). The parameters used in this experiment are: $\lambda = 600$, $\varepsilon_b = 0.05$, $\varepsilon_n = 0.0006$, $\alpha = 0.5$, $\beta = 0.0005$, and $a_{max} = 100$.

In all these models, k-Means, Kohonen SOM and GNG with the proposed approach, it is only necessary to find one nearest neighbor. This is due to the fact that, in Kohonen and GNG, the neighborhood to be updated is determined by connections, and in the case of k-Means, there is no connections and we only update one unit. The same technique could be applied to find more nearest neighbors.

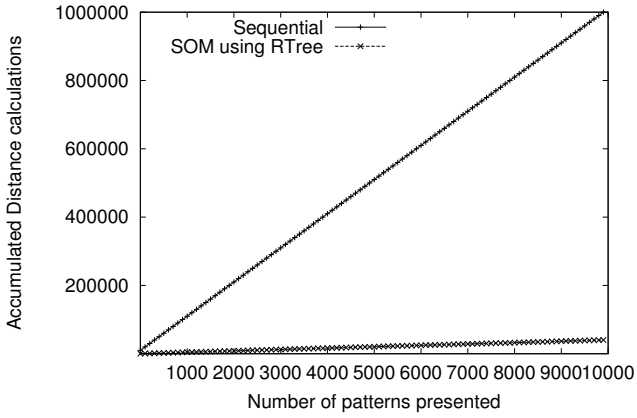
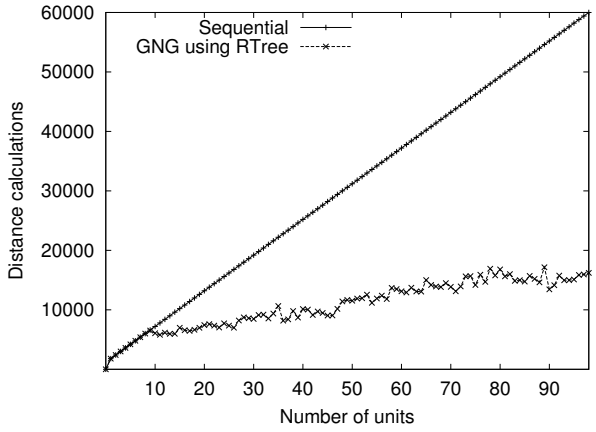


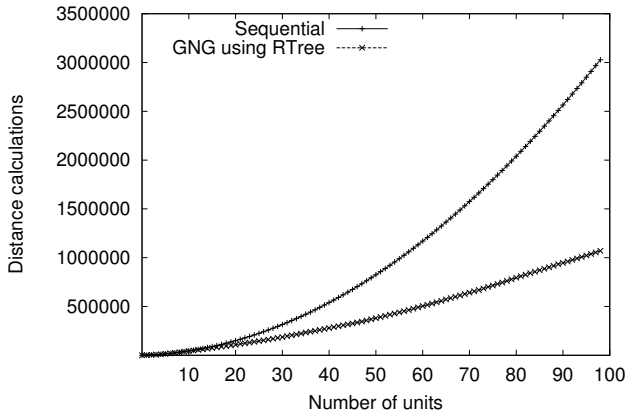
Figure 7: Results obtained by using a SOM Map, online, with Images dataset.

5 Conclusions and Future Works

In this paper, it was showed how Spatial Access Methods can be used to improve the performance of algo-



(a) Distance calculations after a unit has been inserted



(b) Accumulated number of distance calculations

Figure 8: Growing Neural Gas using Images dataset.

rithms which involve comparisons of each of the N points in a dataset with each other point, such as, algorithms of SOM family. Comparative results obtained through the incorporation of R-Trees and k-d Trees into three algorithms of the SOM-family (k-Means clustering, standard SOMs and Growing Neural Gas networks) were presented. The results obtained demonstrate that a speed up of a factor of ten or more, in terms of number of distance computation, can be achieved, what is very significant when large-scale and high dimensional databases are being considered. As a future, we intend to apply more sophisticated SAM to improve the obtained results. By the way, we can also improve the performance obtained by SAM trying to incorporate intelligent techniques to reduce progressively the number of distance calculations involved in the information retrieval process.

Acknowledgements

The authors would like to S. Schönknecht (TU-Berlin) for her help during the experiments, and FAPESP-Brazil (Proc. 99/11835-7) for financial support.

References

[1] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[2] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[3] E. Chavez, G. Navarro, R. Baeza-Yates, and J.L. Marroqun. Proximity searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[4] E. Cuadros-Vargas and R. Francelin Romero. A SAM-SOM Family: Incorporating Spatial Access Methods into Constructive Self-Organizing Maps. In *Proc. IJCNN'02, Intl. Joint Conference on Neural Networks*, pages 1172–1177, Hawaii, HI, 2002. IEEE Press.

[5] C. Faloutsos and K. Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia data. In *Int. Conf. on Management of Data – ACM SIGMOD*, pages 163–174, San Jose, CA, 1994.

[6] B. Fritzke. A growing neural gas networks learns topologies. *Advances in Neural Information Processing Systems*, 7:625–632, 1995.

[7] Alexander G. Gray and Andrew W. Moore. 'N-Body' problems in statistical learning. *Advances in Neural Information Processing Systems*, 13:521–527, 2000.

[8] Antonin Guttman. R-Trees: A dynamic index structure for spatial searching. In *ACM SIGMOD Conference*, pages 47–57, Boston, 1984.

[9] J. MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings - 5th Berkeley Symposium*, 1:281–297, 1967.

[10] J. Moody and C.J. Darken. Fast learning in networks of locally tuned processing units. *Neural Computation*, 1(2):281–294, 1989.

[11] H. Samet. *Spatial Data Structures*. Modern Database Systems. Reading, Addison-Wesley/ACM, kim, w., ed. edition, 1995.

[12] Roberto Figueira Santos-Filho, Agma Traina, C. Traina, and C. Faloutsos. Similarity search without tears: The omni family of all-purpose access methods. *Proceedings of the 17th International Conference on Data Engineering*, pages 623–630, April 2-6 2001. Heidelberg, Germany.

[13] P.N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. *ACM SIGACT-SIAM Symposium on Discrete Algorithms, 4*, pages 311–321, 1993. Austin.