A Self-Adaptive Insert Strategy for Content-Based Multidimensional Database Storage

Sebastian Leuoth, Wolfgang Benn

Department of Computer Science
Chemnitz University of Technology
09107 Chemnitz, Germany

{lese, benn}@cs.tu-chemnitz.de

Abstract

In this paper, we present the current development progress of our dynamic insert strategy based on the Intelligent Cluster Index (ICIx), which is a new type of multidimensional database storage. Opposite to purely value-based interval methods, ICIx performs a semantic clustering of the data objects in a database and keeps the clustering results as basis for storing in a special tree structure (V-Tree). Our paper aims at the quality problem caused by a trade-off between the static clustering that results from the initial training data set and the continuous insertion of data into a database which requires a continuous classification.
The strategy that we propose will solve this problem through a continuous and efficient content-based growing of the initially static clustering. We have developed an additional structure — the C-Tree — which stores the knowlege of the hierarchical clustering component, i.e. hierarchical Growing Neural Gas (GNG), for unsupervised content based classification.
In contrast to other methods (e.g. dynamic versions of R-Trees) we use the C-Tree to process the new tuple. Furthermore, we use a Bayesian approach to determine the degree of adaptation of the knowledge base. Using this value, we update the knowlege base and propagate the resulting changes to the V-Tree. As a result, we obtain a continuous content-based growing.

1 Introduction

Since the emergence of non-standard database systems and their applications, the demand for an efficient processing of multidimensional data increases. Important requests are similarity-based queries such as set queries, range queries, and nearest neighbor queries. Traditional database systems can manage large amounts of data very efficiently, as long as they are ordered and selected by unique attributes in a one-dimensional search space. To close this gap, the Intelligent Cluster Index (ICIx)[3] has been developed.

ICIx is a multidimensional indexing and storage method. Its core idea is to combine methods from the field of Artificial Intelligence — in particular nonparametric classification — with those from the field of database accessing. ICIx uses specific Artificial Neural Networks to classify the data objects in a database, which is considered the multidimensional search space itself [7]. This is done initially by a bulk load step, where the hierarchical clusters of semantically similar data objects are constructed. The clustering procedure uses representative parts or (in the worst case) the complete database as training set. Finally, the resulting hierarchical clustering will be exploited and particular cluster-ids with their associated tuple are transferred into a persistent management tree — the so-called V-Tree. This V-Tree is the actual storage structure. It is used for all supported retrieval operations like point and range queries as well as nearest neighbor queries on the basis of a semantic similarity between the requested data objects. Investigations

have shown that our method is characterized by a fast response time and very few I/O operations in comparison to other methods, such as R*-tree, SS-tree or SR-Tree (for detailed experiments see [4]).

Today, several applications have been explored to prove our approach. The most prominent applications of the ICIx are its use as a database storage engine or as a kind of secondary database index in the form of a set-top box. The database storage engine utilizes our method as primary data organization. The usage as a set-top box is characterized by a loose coupling between our system and an arbitrary existing database system. The query is first sent to the ICIx that processes the multidimensional as well as semantic tasks. Afterwards, the query is transformed in a single primary key selection for which the ICIx returns the key values as response.

Currently, we focus on a flexible space allocation. The hierarchical cluster structure depends heavily on the quality of the training set and may be influenced by the ordering of the tuple. This addiction is tolerable mainly for static data, as it appears in data warehouses or data mining applications. In an online environment or in an environment with frequent input, the efficiency depends on the continuous and stable quality of the clustering — with the problem of its relatively static boundaries. Here, the insertion of a lot of new records can result in very large overlapping regions and a degeneration of the index structure — this problem is known in literature as concept drift [10]. Our current solution uses a hierarchical new-learning-process to compensate such degeneration. In the worst case, this can affect the complete data set. Thus, the aim of this paper is to present an idea for an insert strategy which enables the ICIx to do a continuous, knowledge based self-adaption.

The organization of our paper is as follows: In Section 2, we present an overview of related work in multidimensional data partitioning indexing techniques and their insert strategies. The central Section 3 describes our approach to continuous adaptation. Finally, we conclude with a summary and a description of further research.

2 Related Work

Existing approaches propose a variety of solutions for managing multidimensional data efficiently. A detailed review of these methods can be found in [5]. We focus on the field of data partitioning techniques basing on the R-Tree [6]. These methods propose data distribution as the fundamental basis for a partitioning and can be divided in two classes characterized by their creation method.

1. Dynamic versions are build up by inserting tuples one-by-one. The methods differ in their inserting and splitting strategies. The advantage arises from their continually growing feature. The quality of the resulting structure depends heavily on the order of data records. This methods are susceptible to the concept drift.

2. Static versions are build up during a bulk load step. This enables the methods to create information about the data records so that they can use this information to optimize there structure. The disadvantage of this lies in its read-only usage.

Thus, we were looking for a method to merge these two different streams. We are building an initial version of the ICIx using our Growing Neural Gas (GNG) [2] approach and then switch to a continuous inserting strategy. In contrast to other methods, we rest on the knowledge which was generated in the bulk load step. If we used well known techniques like Linear Node Splitting [1] or Branch Grafting [9], we would destroy the contend-based information and return to a simple rule based acting.

3 The Concept of a Self-Adaptive Insert Strategy

Our approach bases on emphasizing the separation in the construction and insertion phase. We build a new component – the C-Tree (C for Cluster or Control). Its task is to build, adapt, and manage the cluster. During the initial bulk load it applies the data records to the GNG component. After the GNG has finished the growing and classification, the C-Tree stores the internal state and all relevant information. In particular the count and position of the neurons or the edges with their weights are saved. In the next step, the C-Tree propagates the classified tuple to the V-Tree which is used to store the data records. Figure 1 shows this process in a simplified structure of our model. The figure emphasizes the partition in data and query processing component.

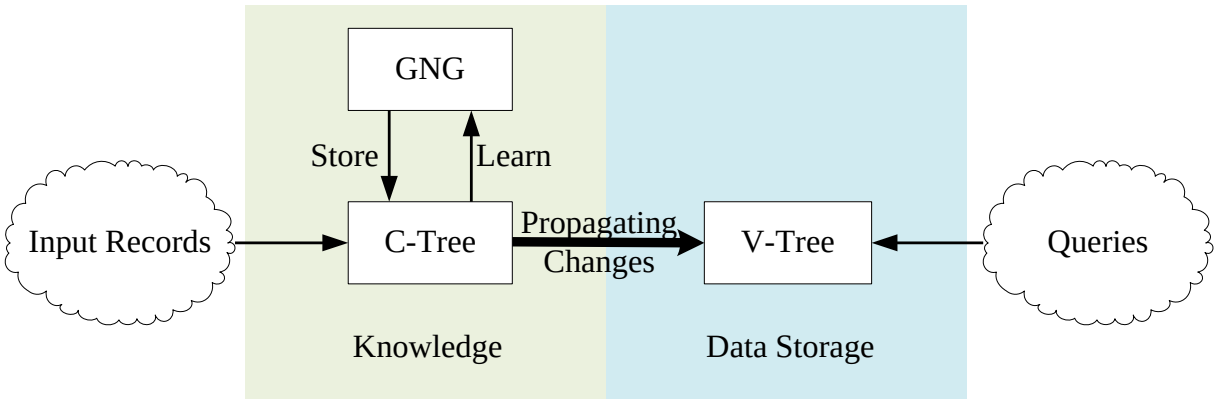

Figure 1: The Figure shows our separation between knowledge and data storage. The growth of the V-Tree is controlled by the C-Tree. Every change in the C-Tree is propagated to the V-Tree.

When a new tuple should be inserted, we can use the C-Tree and its knowledge about the data and their distribution. We use a Bayesian approach to determine the degree of adaptation of the knowledge base. Our insert algorithm (shown in Algorithm 1) starts on top level of the C-Tree and traverses this tree downwards. For each level it uses the neurons to classify the tuple. As in the original GNG procedure, we adapt the neurons. Then we rate the distance between the new tuple and the nearest neuron. If it is greater than a special threshold value, we append a new neuron, if the distance is less we adapt the existent neurons. This strategy corresponds to the Parzen-Windows approach [8]. The size of the volume is determined from the threshold.

The challenge of this method is to handle the resulting changes in a hierarchical environment. If we adapt a neuron on top level, we have to ensure consistency. All classified data records in all following levels still have to be correct. To meet this requirement we use the structure of the C-Tree. Owing to the fact that we have a hierarchy of clusters, we know that the next level again consists of clusters. If we change the position of the neurons in level i, we only have to reclassify the neurons of the all associated levels $i + 1$. There, the adaption effects only a small part of the C-Tree.

Another difficult problem is the overflow handling of nodes in the various levels. As mentioned above, dynamic methods use split strategies to solve this. If we detect an overflow in cluster C on level i, we have to reduce the count of neurons by using an agglomerative clustering method. It selects a group of nearest neurons, e.g. two neurons, and merges it to a new cluster. The resulting cluster will be represented by one new neuron which replaces the others. The position of this new point can be calculated by a centroid or by a center of gravity method which respects the weight of the original neurons. In the next step, we add a new level $i + 1$ into which the

Algorithm 1 Inserting a New Element (`NE`) into the ICIx

```
1: Initialize actual Level of Processing [LP] with root level of the C-Tree
2: Set default value of new Inserting Position [IPos] with blank
3: while [IPos] is blank do
4:    Determine the Minimal Distance [MD] to the Neuron [N] ∈ [LP]
5:    if [MD] ≥ threshold value then
6:       Insert New Neuron [NN]
7:       [IPos] ← New Neuron [NN]
8:    else
9:       Adapt Neuron [N]
10:      if neuron [N] has no child then
11:         Insert NE into value list of neuron [N]
12:         [IPos] ← neuron [N]
13:      else
14:         [LP] ← child level of neuron [N]
15:      end if
16:   end if
17: end while
18: Propagate the changes to the V-Tree
19: Recalculation of the minimum bounding boxes of the V-Tree
```

replaced neurons are moved. The direction of growth of the C-Tree is downwards. If we moved the new neuron up into level $i-1$, we could get two problems. First, in this level another overflow could appear. This could draw up to the root level. If the root level is satisfied, a new root would have to be created. The second problem is the problem of overlapping regions. It is not easy to exclude this possibility. Our proposed direction — down — avoids this problems and the classification of level i will not have to be changed. We only insert one additional level. This situation is shown in Figure 2.

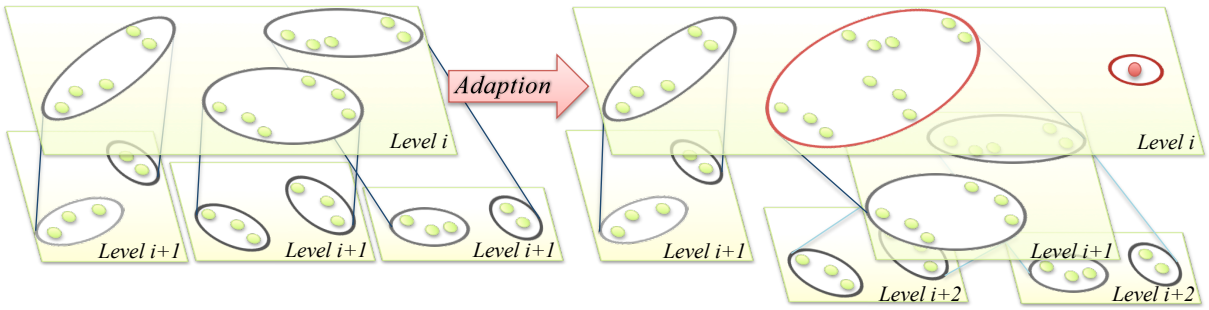

Figure 2: The Figure shows the cluster structure of level i before and after an adaption (highlighted with red). On the right side, a new neuron has been inserted. The number of neurons is limited to 3, so that the two clusters had to be merged.

In all the cases described above, we had to propagate the changes to the V-Tree. However, this operation is not very difficult. We use a link between the clusters of the C-Tree and the corresponding V-Tree nodes to get a fast access. We only have to log the C-Tree changes and finally transmit it to the V-Tree. The only changing effort worth mentioning results from the hierarchical recalculation of the minimum bounding boxes — although this task can occur in every dynamic indexing method.

4 Conclusion and Further Work

We have described a novel self-adaptive insert strategy for content-based multidimensional database storage that rests on a knowledge base. Our strategy introduces a separation between knowledge and data storage. It enables a primarily static method to become a continuous learning method. This is important to reduce the demand for a periodical rebuilding.

In the next step we have to implement the presented algorithm and evaluate our solution. Only then we can compare the different merge, insert and the threshold value strategies by using real and artificial data records. We want to find out the performance of our method and how robust it is against degeneration. In this case, we have to define degeneration and how it can be measured. Another topic is to find out which neurons and classicized tuples are effected by an adaption. We are thinking about a kind of filter which creates a voronoi region. The only neurons that have to be checked are those lying in the same cell.

References

[1] Chuan-Heng Ang and T. C. Tan. New Linear Node Splitting Algorithm for R-trees. In *SSD '97: Proceedings of the 5th International Symposium on Advances in Spatial Databases*, pages 339–349, London, UK, 1997. Springer-Verlag.

[2] Bernd Fritzke. A Growing Neural Gas Network Learns Topologies. In *NIPS*, pages 625–632, 1994.

[3] Sven Gild and Ralf Neubert. Semantische Indexierung mittels dynamische-hierarchischer Neuronaler Netze. Master's thesis, Technische Universität Chemnitz, 1999.

[4] Otmar Görlitz. *Inhaltsorientierte Indexierung auf Basis künstlicher neuronaler Netze.* Shaker, 1st edition, 2005.

[5] Sebastian Leuoth and Wolfgang Benn. Towards SISI – a Self Adaptive Insert Strategy for the Intelligent Cluster Index (ICIx). In *International Conference on Machine Learning and Data Mining MLDM, Leipzig*, 2009.

[6] Yannis Manolopoulos, Alexandros Nanopoulos, Apostolos N. Papadopoulos, and Y. Theodoridis. *R-Trees: Theory and Applications (Advanced Information and Knowledge Processing).* Springer, 1 edition, September 2005.

[7] Ralf Neubert, Otmar Görlitz, and Wolfgang Benn. Towards Content-Related Indexing in Databases. In *Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), 9. GI-Fachtagung, Oldenburg*, pages 305–321. Springer-Verlag, 2001.

[8] Emanuel Parzen. On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.

[9] Thomas Schreck and Zhengxin Chen. R-Tree Implementation Using Branch-Grafting Method. In *SAC '00: Proceedings of the 2000 ACM symposium on Applied computing*, pages 328–332, New York, NY, USA, 2000. ACM.

[10] Gerhard Widmer and Miroslav Kubat. Learning in the Presence of Concept Drift and Hidden Contexts. *Mach. Learn.*, 23(1):69–101, 1996.