Supporting Roll-Up and Drill-Down Operations over OLAP Data Cubes with Continuous Dimensions via Density-Based Hierarchical Clustering

Michelangelo Ceci[1], Alfredo Cuzzocrea[2], and Donato Malerba[1]

[1] Dipartimento di Informatica, Università degli Studi di Bari "Aldo Modo"
via Orabona, 4 - I-70126 Bari - Italy
{ceci,malerba}@di.uniba.it
[2] ICAR-CNR and University of Calabria
Via P. Bucci, 41C, I-87036 Rende, Cosenza, Italy
cuzzocrea@si.deis.unical.it

Abstract. In traditional OLAP systems, roll-up and drill-down operations over data cubes exploit fixed hierarchies defined on discrete attributes that play the roles of dimensions, and operate along them. However, in recent years, a new tendency of considering even continuous attributes as dimensions, hence hierarchical members become continuous accordingly, has emerged mostly due to novel and emerging application scenarios like sensor and data stream management tools. A clear advantage of this emerging approach is that of avoiding the beforehand definition of an ad-hoc discretization hierarchy along each OLAP dimension. Following this latest trend, in this paper we propose a novel method for effectively and efficiently supporting roll-up and drill-down operations over OLAP data cubes with continuous dimensions via a density-based hierarchical clustering algorithm. This algorithm allows us to hierarchically cluster together dimension instances by also taking fact-table measures into account in order to enhance the clustering effect with respect to the possible analysis. Experiments on two well-known multidimensional datasets clearly show the advantages of the proposed solution.

1 Introduction

Traditional OLAP data cubes are defined on top of discrete dimensions that expose fixed hierarchies [4]. To this end, attribute domains of these dimensions are first discretized, and then processed simultaneously in order to obtain the final cube, given a certain measure [4]. Despite this well-consolidated methodology, a recent trend focuses the attention on the problem of effectively and efficiently computing OLAP data cubes defined on top of continuous dimensions (e.g., [5, 7, 6]), as the latter are more suitable to capture real-life dynamics rather than the discrete case. Nevertheless, computing such kind of data cubes poses severe challenges, and several alternatives, such as approximation paradigms (e.g., [7]), have been studied to face-off drawbacks deriving from this challenge. On the other hand, OLAP has also been recognized not only as a "last-stage" technology, but also as an important enabling technology that allows us to enhance the expressive power and the quality of retrieved results of a number of Data Warehousing and Mining techniques (e.g., [2]).

At the convergence of these two relevant challenges of Data Warehousing and Mining research, in this paper we propose and experimentally assess a novel framework, called OLAPBIRCH, whose main goal consists in integrating the clustering algorithm BIRCH [8] and OLAP. This integration permits to boost the benefits of both methodologies into a complex knowledge discovery framework for next-generation applications ranging from analytics to sensor-and-stream data analysis and social network analysis. OLAPBIRCH relies on top of a complex methodology according to which the capability of the clustering algorithm BIRCH are combined with OLAP in order to build a complex hierarchical data structure, called CF-Tree, whose nodes contain clusters retrieved by BIRCH from the target dataset and are organized in an OLAP-like fashion. This permits to exploit all the deriving benefits such as multidimensional and multi-resolution data exploration, roll-up and drill-down operations, interactive exploration, and so forth [4]. Particularly, supporting roll-up and drill-down operations play a significant role, due to the fact that CF-Tree materializes the retrieved clusters at each node, hence this allows us to significantly speed-up the response time due to executing these critical OLAP operations with respect to the baseline case represented by computing new clusters from pre-existent ones at each roll-up (or drill-down) operation. As an important contribution to actual research, OLAPBIRCH considers continuous dimensions instead than classical discrete ones, which is relevant in OLAP (e.g., [5, 7, 6]).

In more details, the proposed approach integrates a revised version of BIRCH in order to obtain a hierarchical organization of dimension instances according to similarity computed both on dimension values (from dimension tables) and measure values (from the fact table). We consider the BIRCH algorithm because, in its original formulation, it shows tree important peculiarities: *i)* Efficiency: The algorithm time complexity is linear in the number of instances to cluster and has a constant space complexity. *ii)* Hierarchical: The algorithm allows us to obtain a hierarchical clustering of instances. *iii)* Incremental: As new instances are given to the algorithm, the hierarchical clustering is revised and adapted by keeping into account memory constraints. All these properties well fit to work with data warehouses, where problems coming from the huge amount of data and require for efficient and incremental solutions. For the specific goal we tackle in this paper, it is also necessary to resort to a hierarchical clustering solution that would permit to give to the OLAP users the opportunity to perform roll-up and drill-down operations over continuous attributes. At this aim, we can exploit the peculiarity of BIRCH that provides high balanced hierarchies that become necessary in OLAP frameworks.

The paper is organized as follows. In the next Section we present the proposed framework OLAPBIRCH. In Section 3, we present an empirical evaluation of the proposed framework an finally, in Section 4 we draw some conclusions.

2 OLAPBIRCH: Combining BIRCH and OLAP

In this section, for the sake of completeness, we first describe the BIRCH algorithm and then we describe proposed modifications that permit to integrate BIRCH in an OLAP framework.

2.1 BIRCH

The BIRCH algorithm [8] works on a hierarchical data structure that the authors call CF tree (Clustering Feature Tree). This data structure permits to partition the incoming data points in an incremental and dynamic way. Each node in the CF tree is called Clustering Feature: Given n data points in a cluster, each of which represented according to a d-size feature vector, CF vector of the cluster is defined as a triple $CF = (n, LS, SS)$, where LS is the linear sum and SS is the square sum of data points. The CF vectors are sufficient to compute information about subclusters like centroid, radius and diameter. They satisfy an important additivity condition, i.e. if $CF_1 = (n_1, LS_1, SS_1)$ and $CF_2 = (n_2, LS_2, SS_2)$ are the clustering features for sets of points S_1 and S_2 respectively, then $CF_1 + CF_2 = (n_1 + n_2, LS_1 + LS_2, SS_1 + SS_2)$ is the clustering feature for the set $S_1 \cup S_2$.

A CF tree is a balanced tree whose structure is similar to that of a B+tree and depends on two parameters: the branching factor B and a user defined threshold T that represents the maximum cluster diameter. Each non-leaf represents a cluster consisting of all the subclusters represented by its entries. In particular, a non-leaf node N_j contains at most B entries of the form $[CF_i, c_i]_{i=1,..,B}$, where c_i is a pointer to the $i-th$ child node of N_j and CF_i is the clustering feature of the cluster identified by c_i. A leaf node contains at most L (typically $L = B$) entries each of the form $[CF_i]$. In the leaves, each node has two pointers $prev$ and $next$ which are used to chain all leaf nodes together. The tree size depends on the T value: the larger the T, the smaller the tree.

The algorithm BIRCH builds a CF tree in four steps. In the first step, BIRCH iteratively receives single data points and builds an initial CF-tree. A point is inserted by inserting the corresponding CF value into the closest leaf. If an entry in the leaf can absorb the new point without violating the threshold T condition, its CF is updated. Otherwise, a new entry is created in the leaf node, and, if the leaf node then contains more than L entries, it and maybe its ancestors are split. In this phase, in order to satisfy RAM constraints, BIRCH frequently rebuilds the whole CF tree by increasing the threshold T and tries to merge as many CF nodes as possible. The rebuild happens sufficiently fast since all needed data is already in RAM. At the same time outliers are removed from the tree and are stored to disk. The algorithm starts with maximum precision at $T = 0$ and as the CF tree grows larger than the available memory, it iteratively tries to find suitable cluster sizes by increasing T to be larger than the smallest distance between two entries in the tree. In the second step, the algorithm condenses the CF tree to a desirable size depending on the clustering algorithm employed in step three. This can involve removing outliers and further merging of clusters. In the third step, the algorithm employs a *global* clustering algorithm using the CF tree's leaves as input. This step, as claimed in [8], permits to avoid the undesirable effect of the skewed input order, and splitting triggered by space constraints. In this phase, the CF vectors allow for effective distance metrics computation. In the last step, a labeling procedure is performed. This means that, if desired, the actual data points can be associated with the generated clusters.

In the implementation we provide, the used distance measure is the variance increase distance [8] defined as follows:

Definition 1 (Variance Increase Distance).
Let C_1 and C_2 be two clusters, where $C_1 = \{x_i\}_{i=1..n_1}$ and $C_2 = \{x_i\}_{i=n_1+1..n_2}$. The variance increase distance between C_1 and C_2 is defined as:

$$D = \sum_{k=1}^{n_1+n_2} \left(x_k - \frac{\sum_{l=1}^{n_1+n_2} x_l}{n_1+n_2} \right)^2 - \sum_{i=1}^{n_1} \left(x_i - \frac{\sum_{l=1}^{n_1} x_l}{n_1} \right)^2 - \sum_{i=n_1+1}^{n_2} \left(x_i - \frac{\sum_{l=n_1+1}^{n_2} x_l}{n_2} \right)^2$$

In our implementation, the clustering algorithm for the third step is the well-known DBSCAN [3] algorithm that performs a density based clustering. Density based clustering is performed on cluster centroids (that can be easily computed from the CF vectors and represent aggregated data) and allows us to further aggregate data that show similar peculiarities.

2.2 OLAPBIRCH

The integration of the implemented BIRCH algorithm in the OLAP solution we present, is not a trivial task since different issues have to be considered: First, OLAP queries can consider all the levels of the hierarchy and not only the last level. This means that it is necessary to have refined clusters not only in the last level of the hierarchy, but also in intermediate levels. Second, in OLAP frameworks, the user is typically able to control size of hierarchies, but this is not possible in the original BIRCH algorithm. Third, although the last step of the BIRCH algorithm is not mandatory, this step is necessary in our framework in order to simplify the computation of OLAP queries. Fourth, in order to avoid the combinatorial explosion that is typical in multidimensional clustering, it is necessary to focus only on interesting continuous dimension attributes.

In order to face with the first issue, we revised the clustering algorithm in order to allow the system to run the *global* clustering algorithm also in intermediate nodes of the tree. At this purpose, we extended the CF tree structure by providing pointers $prev$ and $next$ to each internal node. This allows us to linearly scan a each single level of the tree. In Figure 1, we report a graphical representation of the CF tree structure used in the proposed framework.

As for the second issue, in addition to the memory space constraints, we consider also an additional constraint that forces tree rebuilding when a maximum number of levels (MAX_LEV) is exceeded. This is coherent with the goal of having a limited number of levels, as in classical OLAP systems.

As for the third issue, given the maximum number of levels MAX_LEV and the branching factor B, it is possible to use a numerical representation of the complete path of clusters for each dimension instance so that the classical B+tree index structure can be used in order to allow efficient computation of range queries[5]. The representation is in the form $< d_1 d_2 \dots d_{MAX\_LEV} >$, where each d_i is a sequence of $\ulcorner log_2 B \urcorner$ bits that permits to identify each subcluster. The number obtained in this way is then used to perform roll-up and drill-down operations.

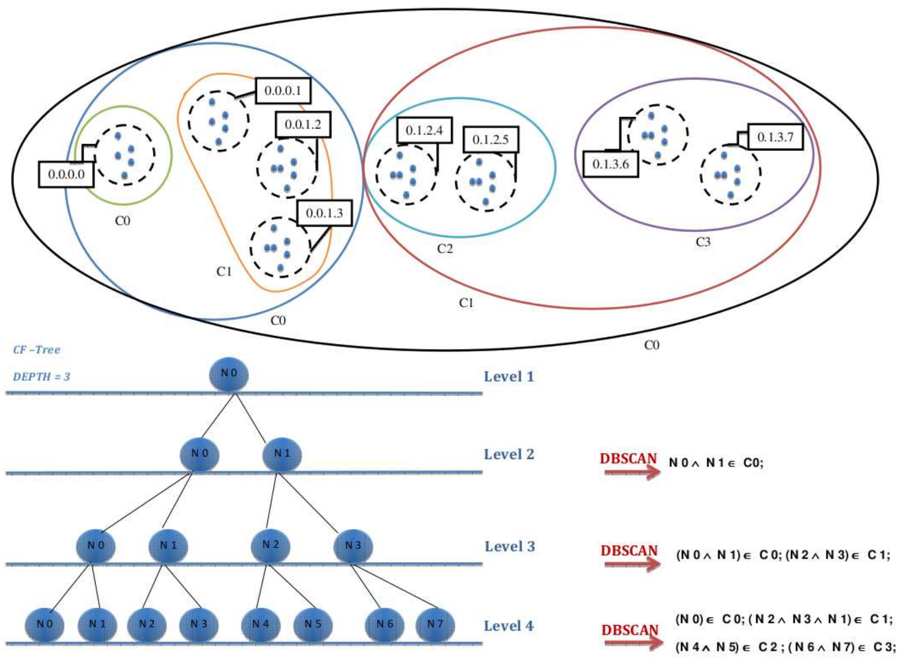

Fig. 1. OLAPBIRCH: an example of CF tree.

Finally, as for the fourth issue, in order to integrate the algorithm in an OLAP framework, we defined a language that permits to specify the attributes to be considered in the clustering phase. At this purpose, we have exploited the Mondrian[3] project that permits to represent a multidimensional schema of a data warehouse by means of an XML file. This file permits to define a mapping between the multidimensional schema and tables and attributes stored in the database. Main elements in this XML file are: the data source, cubes, measures, the fact table, dimensions and hierarchies.

For our purposes, we have modified the DTD in order to allow different types of hierarchies. The modified portion of the DTD is:

```
<!ELEMENT Hierarchy ((\%Relation;)?,(Level)∗,
        (MemberReaderParameter)∗,(Attribute)+, (Depth))>
   <!ATTLIST Hierarchy
        hasAll (true|false) #REQUIRED
        allMemberName CDATA #IMPLIED
        allMemberCaption CDATA #IMPLIED
        primaryKey CDATA #IMPLIED
        primaryKeyTable CDATA #IMPLIED
```

[3] http://sourceforge.net/projects/mondrian/files/mondrian/

```
          defaultMember CDATA #IMPLIED
          memberReaderClass CDATA #IMPLIED>
<!ELEMENT Attribute EMPTY>
    <!ATTLIST Attribute
          name CDATA #IMPLIED
          table CDATA #REQUIRED
          column CDATA #REQUIRED
          nameColumn CDATA #REQUIRED
          type (Numeric) Numeric #REQUIRED>
<!ELEMENT Depth EMPTY>
    <!ATTLIST Depth value (Numeric) Numeric #REQUIRED>
```

The DTD so modified permits to add two new elements ($< Attribute >$ and $< Depth >$) to the elements defined in $< Hierarchy >$. The $< Attribute >$ element permits to define one or more attributes to be used in the clustering procedure. Properties that can be defined in the $< Attribute >$ tag are: name - attribute name; table - table that contains the attribute; column: database column name; nameColumn: database column name (alias); type: SQL attribute type. The $< Depth >$ element permits to specify the maximum depth of the CF-tree.

The CF-tree is updated when a new dimension tuple is saved in the data warehouse while DBSCAN is run only when OLAP queries are executed and clusters are not updated. This permits to focus our attention only to levels that are actually used in the queries. It is noteworthy that, differently from [7], the global clustering is run on compact representations of data and does not pose efficiency problems.

Example 1. Le us consider the database schema reported in Figure 2 where $lineitem$ is the fact table and $orders$ is a dimensional table. By selecting, in the XML file, the attributes $orders.o\_totalprice$ and $orders.o\_orderpriority$:

```
< Attribute name="totalprice" table="orders" column="
        o_totalprice" nameColumn="o_totalprice"
        type="Integer"/>
< Attribute name="orderpriority" table="orders" column="
        o_orderpriority" nameColumn="o_orderpriority"
        type="Integer" />
< Depth value="20"/>
```

we have that the OLAP engine performs clustering on the following database view:

```
SELECT l_quantity, l_extendedprice, l_discount, l_tax,
          o_totalprice, o_orderpriority
FROM lineitem, orders
WHERE l_orderkey = o_orderkey
```

3 Experimental Evaluation and Analysis

In order to evaluate the effectiveness of the proposed solution, we performed experiments on two real world datasets. The first dataset is the SPAETH Cluster Analysis

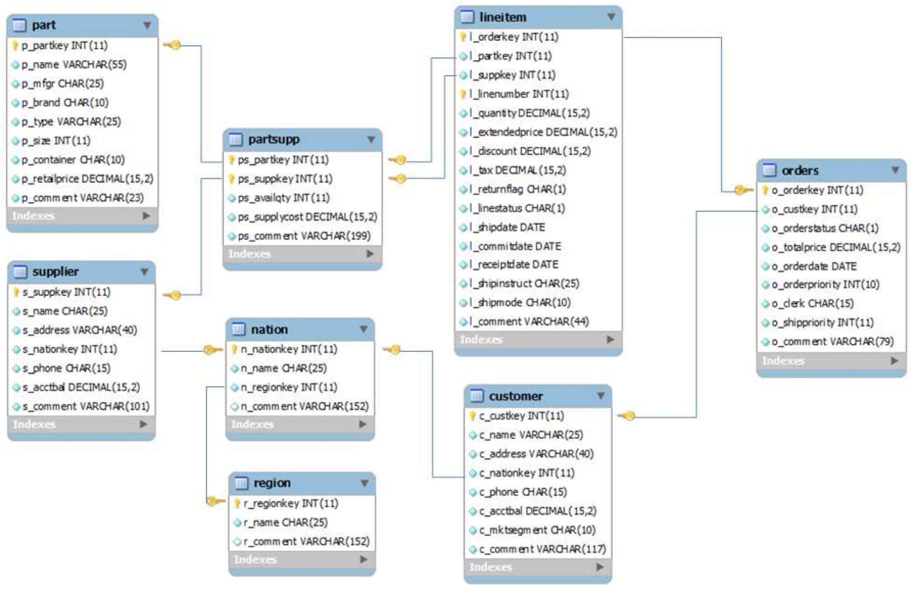

Fig. 2. TPC-H database schema

Datasets[4], a small dataset that allows us to visually evaluate the quality of extracted clusters. The second dataset is the well-know TPC-H benchmark (version 2.1.0)[5]. In Figure 2 we report the relational schema of TPC-H implemented on PostgreSQL, which we used as supporting DBMS. The TPC-H database holds data about the ordering and selling activities of a large-scale business company. For experiments we used the 1GB version of TPC-H [1] containing more that 1×10^6 tuples in the fact table. On this last dataset, we performed experiments on the scalability on the algorithm and we collected results in terms of running times and cluster quality. Clustering is performed on the fact table measures as well as on the the attributes orders.o_totalprice and orders.o_orderpriority as specified in Example 1. In order to force the OLAPBIRCH framework to work in the worst case scenario, running times are obtained by forcing the system to work when the examples are given one by one. The cluster quality is measured according to the *weighted average cluster diameter square measure*:

$$Q = \sum_{i=1..K} n_i(n_i-1)D_i^2 / \sum_{i=1..K} n_i(n_i-1) \tag{1}$$

where K is the number of obtained clusters, n_i is the cardinality of the i-th cluster and D_i is the diameter of the i-th cluster. The smaller the Q value, the higher the cluster quality.

In Figure 3, we report a graphical representation of obtained clusters. As we can see, the global clustering (DBSCAN) is necessary in order to have good quality clus-

[4] http://people.sc.fsu.edu/∼jburkardt/datasets/spaeth/spaeth.html
[5] Transactions Processing Council Benchmarks. Available from: http://www.tpc.org.

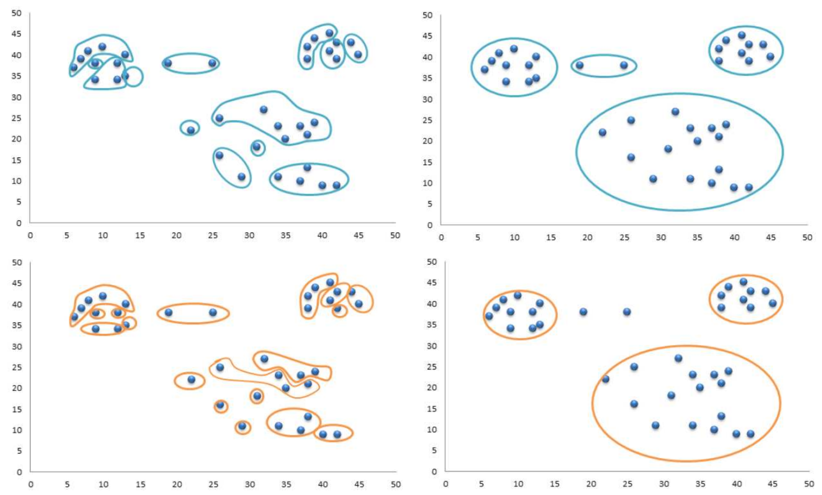

Fig. 3. Clustering effect on Spaeth dataset. CF-tree is obtained with B=L=2. Left: OLAPBIRCH without DBSCAN, Right: OLAPBIRCH with DBSCAN; Top: $LEVEL = 6$, Bottom: $LEVEL = 7$. Points outside clusters are considered outliers.

No of points	Running time (s)	Q	No of rebuilds
60×10^3	776	0.08	5
100×10^3	1819	0.07	5
500×10^3	41,646	0.018	5
1.1×10^6	172,800	0.039	9

Table 1. TPC-H: scalability results. MAX_LEV= 20, $B = L = 2$

ters. Moreover, by increasing the depth of the tree, it is possible to have more detailed clusters.

Results obtained on the TPC-H database are reported in Table 1. From these results, it is possible to see that the quality of the clusters does not deteriorate when the number of examples increases. We can also see that the number of times that the CF-tree is rebuilt is very small, even for huge datasets.

Figure 4 shows a different perspective of the obtained results. In particular, it shows that there is strong correlation between the region dimension (that is not considered during the clustering phase) and the obtained clusters. This means that numerical properties of the orders change in distribution between the regions where the order is performed.

4 Conclusions and Future Work

In this paper we have presented the framework OLAPBIRCH. This framework integrates a clustering algorithm in an OLAP engine in order to support roll-up and rill-down operations on numerical dimensions. OLAPBIRCH integrates a revised version of the BIRCH clustering algorithm that permits to incrementally revise hierarchical

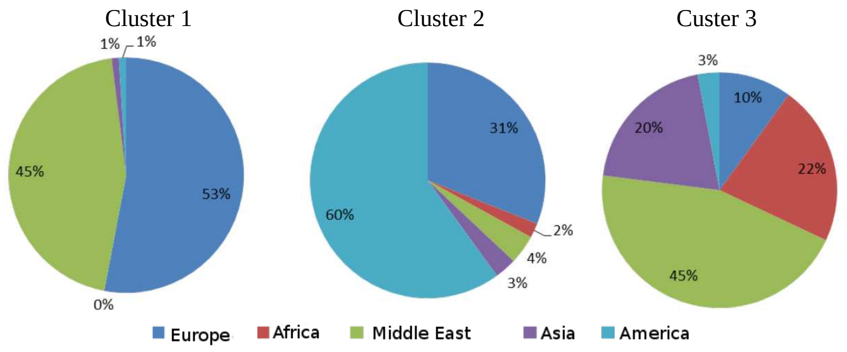

Fig. 4. TPC-H: Data distribution over the Region dimension.

clustering for all the levels of the hierarchy. Preliminary results show the effectiveness of the proposed solution on large real world datasets. For future work we intend to compare our framework with competitive frameworks and prove its applicability in answering to range queries and we intend to run experiments by varying input parameters in order to give better insights on their definition.

Acknowledgment

This work is partial fulfillment of the research objective of ATENEO-2009 project "Estrazione, Rappresentazione e Analisi di Dati Complessi".

References

1. A. Cuzzocrea. Improving range-sum query evaluation on data cubes via polynomial approximation. *Data Knowl. Eng.*, 56(2):85–121, 2006.
2. A. Cuzzocrea and P. Serafino. Clustcube: An olap-based framework for clustering and mining complex database objects. In *SAC*, 2011.
3. M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231, 1996.
4. J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub totals. *Data Min. Knowl. Discov.*, 1(1):29–53, 1997.
5. D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi. Selectivity estimators for multidimensional range queries over real attributes. *VLDB J.*, 14(2):137–154, 2005.
6. N. Karayannidis and T. K. Sellis. Hierarchical clustering for olap: the cube file approach. *VLDB J.*, 17(4):621–655, 2008.
7. J. Shanmugasundaram, U. M. Fayyad, and P. S. Bradley. Compressed data cubes for olap aggregate query approximation on continuous dimensions. In *KDD*, pages 223–232, 1999.
8. T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In H. V. Jagadish and I. S. Mumick, editors, *SIGMOD Conference*, pages 103–114. ACM Press, 1996.