
Spatial Aggregation Query

DONGHUI ZHANG
College of Computer & Information Science
Northeastern University

Synonyms

spatial aggregate computation

Definition

Given a set O of weighted point objects and a rectangular query region r in the d -dimensional space, the **spatial aggregation query** asks the total weight of all objects in O which are contained in r .

This query corresponds to the SUM aggregation. The COUNT aggregation, which asks for the number of objects in the query region, is a special case when every object has equal weight.

The problem can actually be reduced to a special case, called the **dominance-sum query**. An object o_1 dominates another object o_2 if o_1 has larger value in all dimensions. The dominance-sum query asks for the total weight of objects dominated by a given point p . It is a special case of the spatial aggregation query, when the query region is described by two extreme points: the lower-left corner of space, and p .

The spatial aggregation query can be reduced to the dominance-sum query in the 2D space, as illustrated below. Given a query region r (a 2D rectangle), let the four corners of r be *lowerleft*, *upperleft*, *lowerright*, and *upperright*. It is not hard to verify that the spatial aggregate regarding to r is equal to

$$\begin{aligned} & \text{dominancesum}(\text{upperright}) \\ - & \text{dominancesum}(\text{lowerright}) \\ - & \text{dominancesum}(\text{upperleft}) \\ + & \text{dominancesum}(\text{lowerleft}) \end{aligned}$$

Historical Background

In computational geometry, to answer the dominance-sum query, an *in-memory* and *static* data structure called the **ECDF-tree** [2] can be used. The ECDF-tree is a *multi-level data structure*, where each level corresponds to a different dimension. At the first level (also called *main branch*), the d -dimensional ECDF-tree is a full binary search tree whose leaves store the data points, ordered by their position in the first dimension. Each internal node of this binary search tree stores a *border* for all the points in the left sub-tree. The *border* is itself a $(d-1)$ -dimensional ECDF-tree; here points are ordered by their positions in the second dimension. The collection of all these border trees forms the second level of the structure. Their respective borders are $(d-2)$ -dimensional ECDF-trees (using the third dimension and so on). To

answer a dominance-sum query for point $p = (p_1, \dots, p_d)$, the search starts with the root of the first level ECDF-tree. If p_1 is in the left sub-tree, the search continues recursively on the left sub-tree. Otherwise, two queries are performed, one on the right sub-tree and the other on the border; the respective results are then added together.

In the fields of GIS and spatial databases, one seeks for *disk-based* and *dynamically updateable* index structures. An approach is to externalize and dynamize the ECDF-tree. To *dynamize* a static data structure some standard techniques can be used [4]. For example, the *global rebuilding* [8] or the *logarithmic method* [3]. To *externalize* an internal-memory data structure, a widely used method is to augment it with block-access capabilities [11]. Unfortunately, this approach is either very expensive in query cost, or very expensive in index size and update cost.

Another approach to solve the spatial aggregation query is to index the data objects with a multi-dimensional access method like the *R*-tree* [1]. The R*-tree (and the other variations of the R-tree) clusters nearby objects into the same disk page. An index entry is used to reference each disk page. Each index entry stores the *minimum bounding rectangle (MBR)* of objects in the corresponding disk page. The index entries are then recursively clustered based on proximity as well. Such multi-dimensional access methods provide efficient *range* query performance in that sub-trees whose MBRs do not intersect the query region can be pruned. The spatial aggregation query can be reduce to the range search: retrieve the objects in the query region and aggregate their weights on-the-fly. Unfortunately, when the query region is large, the query performance is poor.

An optimization proposed by [7, 9] is to store, along with each index entry, the total weight of objects in the referenced sub-tree. The index is called the aggregate R-tree, or *aR-tree* in short. Such aggregate information can improve the aggregation query performance is that if the query region fully contains the MBR of some index entry, the total weight stored along with the index entry contributes to the answer, while the sub-tree itself does not need to be examined. However, even with this optimization the query effort is still affected by the size of the query region.

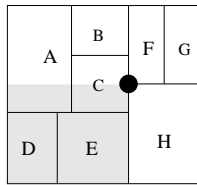
Scientific Fundamentals

This section presents a better index for the dominance-sum query (and in turn the spatial aggregation query) called the *Box-Aggregation Tree*, or **BA-tree** in short.

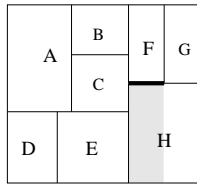
The BA-tree is an augmented k-d-B-tree [10]. The k-d-B-tree is a disk-based index structure for multi-dimensional point objects. Unlike the R-tree, the k-d-B-tree indexes the whole space. Initially, when there are only a few objects, the k-d-B-tree uses a single disk page to store them. The page is responsible for the whole space in the sense that any new object, wherever it is

located in space, should be inserted to this page. When the page overflows, it is split into two using a hyperplane corresponding to a single dimension. For instance, order all objects based on dimension one, and move the half of the objects with larger dimension-one values to a new page. Each of these two disk pages are referenced by an index entry, which contains a *box*: the space the page is responsible for. The two index entries are stored in a newly created index page. As more split happens, the index page contains more index entries.

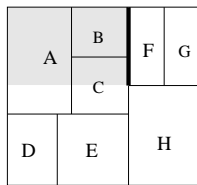
For ease of understanding let's focus discussion on the 2D space. Figure 1 shows an exemplary index page of a BA-tree in the 2D space. As in the k-d-B-tree, each index record is associated with a *box* and a *child* pointer. The boxes of records in a page do not intersect and their union creates the box of the page.



(a) points affecting the subtotal of F



(b) points affecting the x-border of F



(c) points affecting the y-border of F

Spatial Aggregation Query. Figure 1 The BA-tree is a k-d-B-tree with augmented border information.

As done in the ECDE-tree, each index record in the k-d-B-tree can be augmented with some *border* information. The goal is that a dominance-sum query can be answered by following a single sub-tree (in the main branch). Suppose in Figure 1a, there is a query point contained in the box of record *F*. The points that may affect the dominance-sum query of a query point in *F.box* are those dominated by the upper-right point of *F.box*. Such points belong in four groups: (1) the points contained in *F.box*; (2) the points dominated by the low point of *F* (in the shadowed region of Figure 1a); (3) the points below the lower edge of *F.box* (Figure 1b); and (4) the points to the left of the

left edge of *F.box* (Figure 1c).

To compute the dominance-sum for points in the first group, a recursive traversal of $subtree(F)$ is performed. For points in the second group, in record *F* a single value (called *subtotal*) is kept, which is the total value of all these points. For computing the dominance-sum in the third group, an *x-border* is kept in *F* which contains the *x* positions and values of all these points. This dominance-sum is then reduced to a 1D dominance-sum query for the border. It is then sufficient to maintain these *x* positions in a 1D BA-tree. Similarly, for the points in the fourth group, a *y-border* is kept which is a 1D BA-tree for the *y* positions of the group's points.

To summarize, the 2D BA-tree is a k-d-B-tree where each index record is augmented with a single value *subtotal* and two 1D BA-trees called *x-border* and *y-border*, respectively. The computation for a dominance-sum query at point *p* starts at the root page *R*. If *R* is an index node, it locates the record *r* in *R* whose box contains *p*. A 1D dominance-sum query is performed on the *x-border* of *r* regarding *p.x*. A 1D dominance-sum query is performed on the *y-border* of *r* regarding *p.y*. A 2D dominance-sum query is performed recursively on $page(r.child)$. The final query result is the sum of these three query results plus *r.subtotal*.

The insertion of a point *p* with value *v* starts at the root *R*. For each record *r* where *r.lowpoint* dominates *p*, *v* is added to *r.subtotal*. For each *r* where *p* is below the *x-border* of *r*, position *p.x* and value *v* are added to the *x-border*. For each record *r* where *p* is to the left of the *y-border* of *r*, position *p.y* and value *v* are added to the *y-border*. Finally, for the record *r* whose box contains *p*, *p* and *v* are inserted in the $subtree(r.child)$. When the insertion reaches a leaf page *L*, a leaf record that contains point *p* and value *v* is stored in *L*. Since the BA-tree aims at storing only the aggregate information, not the objects themselves, there are chances where the points inserted are not actually stored in the index, thus saving storage space. For instance, if a point to be inserted falls on some border of an index record, there is no need to insert the point into the sub-tree at all. Instead, it is simply kept in the border that it falls on. If the point to be inserted falls on the low point of an internal node, there is even no need to insert it in the border; rather, the *subtotal* value of the record is updated.

The BA-tree extends to higher dimensions in a straightforward manner: a *d*-dimensional BA-tree is a k-d-B-tree where each index record is augmented with one *subtotal* value and *d* borders, each of which is a (*d*-1)-dimensional BA-tree.

Key Applications

One key application of efficient algorithms for the spatial aggregation query is interactive GIS systems. Imagine a user interacting with such a system. She sees a map on the computer screen. Using the mouse, she can select a rectangular region on the map. The screen

zooms in to the selected region. Besides, some statistics about the selected region, e.g. the total number of hotels, total number of residents, and so on, can be quickly computed and displayed on the side.

Another key application is in data mining, in particular, to compute *range-sums* over data cubes. Given a d -dimensional array A and a query range q , the range-sum query asks for the total value of all cells of A in range q . It is a crucial query for online analytical processing (OLAP). The best known in-memory solutions for data cube range-sum appear in [6, 5]. When applied to this problem, the BA-tree differs from [6] in two ways. First, it is disk-based, while [6] presents a main-memory structure. Second, the BA-tree partitions the space based on the data distribution while [6] does partitioning based on a uniform grid.

Future Directions

The update algorithm for the BA-tree is omitted from here, but can be found in [12]. Also discussed in [12] are more general queries, such as spatial aggregation over objects with extent.

The BA-tree assumes that the query region is an axis-parallel box. One practical direction of extending the solution is to handle arbitrary query regions, in particular, polygonal query regions.

Cross References

1. Progressive Approximate Aggregate Queries
2. Spatial OLAP

Recommended Reading

- [1] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *SIGMOD*, pages 322–331, 1990.
- [2] J. L. Bentley. Multidimensional Divide-and-Conquer. *Communications of the ACM*, 23(4):214–229, 1980.
- [3] J. L. Bentley and N. B. Saxe. Decomposable Searching Problems I: Static-to-Dynamic Transformations. *Journal of Algorithms*, 1(4):301–358, 1980.
- [4] Y. Chiang and R. Tamassia. Dynamic Algorithms in Computational Geometry. *Proc. of the IEEE, Special Issue on Computational Geometry*, 80(9):1412–1434, 1992.
- [5] C. Chung, S. Chun, J. Lee, and S. Lee. Dynamic Update Cube for Range-Sum Queries. In *VLDB*, pages 521–530, 2001.
- [6] S. Geffner, D. Agrawal, and A. El Abbadi. The Dynamic Data Cube. In *EDBT*, pages 237–253, 2000.
- [7] I. Lazaridis and S. Mehrotra. Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure. In *SIGMOD*, pages 401–412, 2001.
- [8] M. H. Overmars. The Design of Dynamic Data Structures. In *Lecture Notes in Computer Science 156*, 1983.
- [9] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient OLAP Operations in Spatial Data Warehouses. In *Proc. of Symposium on Spatial and Temporal Databases (SSTD)*, pages 443–459, 2001.
- [10] J. Robinson. The K-D-B Tree: A Search Structure For Large Multidimensional Dynamic Indexes. In *SIGMOD*, pages 10–18, 1981.
- [11] J. S. Vitter. External Memory Algorithms and Data Structures. *ACM Computing Surveys*, 33(2):209–271, 2001.
- [12] D. Zhang, V. J. Tsotras, and D. Gunopulos. Efficient Aggregation over Objects with Extent. In *PODS*, pages 121–132, 2002.