APPLIED
INFORMATICS

Innsbruck, Austria
February 20-22, 1996

Editor: M. H. Hamza

A Publication of
The International Association of Science
and Technology for Development - IASTED

ISBN: 0-88986-195-1
Efficient Processing of Spatial Selection and Join in Databases

Azzam Ibrahim¹, and Farshad Fotouhi²

¹ Metlife - MetSource Consulting, 200 Galleria Oficcenter, Suite 400
Southfield, Michigan 48034

² Department of Computer Science- Wayne State University, Detroit, MI 48202, USA

Keywords
SB+ -Tree, Spatial Selection, Spatial Join, R-Tree.

Abstract
A spatial database allows for storing and retrieving spatial objects and consists of spatial relations which may contain spatial as well as non-spatial data. Spatial data can be points, lines, rectangles, surfaces, or more complex objects. Similar to other database systems, efficient query processing relies upon auxiliary data structures used to support spatial indexing of these objects. In this paper, we discuss how the SB+-tree can be used to process the spatial selection and join operations: Point-Containment and Regions-Containment respectively.

1. Introduction

The B+-tree has proven to be an efficient data structure for indexing one-dimensional data [1]. It is a paged-secondary memory access structure that has been used by most commercial textual database management systems. R-tree is another paged-secondary memory access structure for multidimensional spaces. However, for most versions of this structure, multiple path traversal is required in order to process a query.

We have been driven by the notion of modifying the B+-tree so that it can be used as a multi-attribute indexing mechanism without the need to transform the native space to one-dimension. This has led us to introduce the "Spatial" B+-tree (SB+-tree) which is a dynamic multilevel balanced tree that can be used to access point as well as region objects. Moreover, it guarantees that only one path needs to be followed to process a query. In addition, it efficiently processes the join operators defined in the previous section.

SB+-Tree: Spatial B+-Tree

The SB+-tree is a balanced point and region access structure. Each dimension of the space is indexed by an independent SB+-tree. When a query is processed, only the SB+-trees corresponding to the dimensions referenced by the query are searched and the output is produced in terms of the outcome of the individual searches. The following two definitions explain the terms: Indexing Point and Indexing Set.

Definition
The point ip is an indexing point of space S with respect to dimension p if there exists some object O in S such that the line p=ip represents the lower or upper bounding line of O with respect to pin S.

Definition
The indexing set of dimension p in a particular space; denoted by IPp is the set of all indexing points of p in that space.

For each indexing set of an axis, we create an SB+-tree with degree n (as in B+-tree). Intermediate node entries are of the form: <P1, ip1, P2, ip2, ..., Pn-1, ipn-1, Pn>, where Pj, 1 ≤ j ≤ n, is the subtree pointer; ipj ∈ IPp for 1 ≤ j ≤ n-1, is an indexing point and ip1 < ip2 < ... < ipn-1. Leaf node entries are of the form: <(ip1, Pr1), (ip2, Pr2), ..., (ipn-1, Prn-1), Pnext, Pprevious>, where, Pnext and Pprevious are pointers to the next and previous leaf nodes respectively. ip1 < ip2 < ... < ipn-1, and Prj is a data pointer to the data block associated with ipj. The data block contains tuples of the form (objid, rel, status, pr) where objid is the MBR id of an object in relation rel such that the object occurs at ipj with respect to the axis of the SB+-tree. The object is pointed to by pointer pr. If the object is a point data in the space status = 'p'. For the given axis, status = 'b' indicates that the object is a region beginning at ipj; status = 'e' indicates that the region ends at ipj; status = 'c' implies that the region starts at some point before ipj and continues to occur until some point after ipj. For non-zero size objects the MBR could start, end or continue at ipj; whereas in case of point data it is the location of the point on the corresponding dimension. Each data block ipj.datablock is associated with two pointers ipj.next and ipj.previous that point to the data blocks associated with the previous and next indexing.
between the data blocks associated with two indexing points \( i_p \text{ and } i_p' \) produces all objects that extend and cover the range \([i_p..i_p']\). The \( SB^+ \)-tree of the Y-axis is used with respect to point \( y \) and another set of objects is retrieved. Data objects that are common between the two sets represent the objects containing the point \((x,y)\) in the space. The following algorithm formalizes the previous procedure.

**Algorithm : Point-Containment in \( R \)**

**Input** : The root of an \( SB^+ \)-tree, and point’s coordinate with respect to the corresponding axis: \( a \).

**Assumption**: \( i_p \text{min} < a < i_p \text{max} \)

**Output** : Objects of \( R \) that contain \( a \) for the relevant axis.

1. \( Oaxis = \{ \} \) /* initializing the output set */
2. Search for the leaf node that has an entry \( i_p = a \)
3. If (there is an entry such that \( i_p = a \))
   
   for (every \((r_i, R, \text{status, } pr) \in i_p \text{.datablock})

   \( Oaxis = Oaxis \cup r_i \);  
4. else
   
   4.1 Let \( i_p = \) the indexing point immediately preceding \( c_i = \) on the left,
   4.2 \( i_p = \) the indexing point immediately following \( c_i = \) on the right.
   // \( i_p / i_p \) could be in the same leaf node as \( c_i \) or in // the previous/next leaf node.
   4.3 for (every \((r_i, R, \text{status, } pr) \in (i_p \text{.datablock } \cap
   \text{nip.datablock}))

   \( Oaxis = Oaxis \cup r_i \);  

   return \( Oaxis \)

2. **Point-Containment Using \( SB^+ \)-Tree**

   The \( SB^+ \)-tree can be used to find non-zero size objects of a specific type say \( R \) that spatially contain a particular point \((x,y)\). The process is conducted by first using the \( SB^+ \)-tree of the X-axis to find the indexing point \( i_p \) that has a value equal to \( x \). The data block of \( i_p \) is searched and data objects of type \( R \) are retrieved. However, if there is no \( i_p \) with the value \( x \), data blocks of the indexing points preceding and following \( x \) are intersected and objects of type \( R \) are retrieved. The intersection

3. **Regions-Containment Using \( SB^+ \)-Tree**

   This operator can be conducted using the \( SB^+ \)-trees associated with the space. In order for an object \( r \) to contain object \( s \) in the space, the extension of the \( s \)’s MBR must be a sub interval of that of \( r \) with respect to every dimension in the space. The algorithm for finding all objects of \( R \) that contain objects of \( S \) with respect to a particular dimension is given below. The algorithm requires two data structure \( ArrR \) (length is size of \( R \)) and \( ArrS \) (length is size of \( S \)). Every object of \( R \) has a
corresponding entry in ArrR of two fields. ArrR[ri].flag=1 denotes that the beginning of ri.mbr with respect to the corresponding axis has been encountered while the indexing points are being traversed from left to right. ArrR[ri].flag=0 means that either the beginning of ri.mbr with respect to the relevant axis has not been traversed or it has been traversed but its ending is yet to be traversed. ArrS is set in the same way with respect to objects of S. Once ArrR[ri].flag is set to 1, every object of S starting afterwards is added to the set ArrR[ri].SObjects. When the indexing point corresponding to the end of the MBR of ri is visited all objects of S in ArrR[ri].SObjects are checked using ArrS to find the objects that have already ended on or before this point. Those objects of S are contained by ri for the relevant axis.

Algorithm: Regions-Containment Join of relations R and S.
Input: SB+-tree, and object types.
Output: \{(ri,sj): ri \in R, sj \in S and ri contains sj with respect to the relevant axis\}.
Assumption: R and S objects are regions in the space.
Local Structures:
ArrR is an array[rmin..rmax] of ArrRElement;
ArrS is an array[smin..smax] of ArrSElement;
ArrRElement is a structure of
   \{flag is binary; l/flag= 1 => object has started.
   flag=O => object has ended
   SObjects is a set of objects of type S;\}
ArrSElement is a structure of
   \{flag is binary; //flag=1 => object has started.
   flag=O => object has ended \}

1. Oaxis= {};
2. for (every ri \in [rmin..rmax])
   \{ ArrR[ri].flag=0;
   ArrR[ri].SObjects= {} \; \}
3. for (every sj \in [smin..smax]) ArrS[sj].flag=0;
4. Start with the leftmost leaf node of the SB+-tree;
5. for (ip=ipmin; ip<= ipmax; next ip){
   5.1 for (every (ri,R,‘b’,pr) \in ip.datablock)
      ArrR[ri].flag= 1;
      for (every (sj,S,status,pr) \in ip.datablock){
         if (status== ‘b’)
         \{ ArrS[sj].flag= 1;
         for (every ri such that
            ArrR[ri].flag=1)
            ArrR[ri].SObjects= ArrR[ri].SObjects \cup \{sj\};
         \} else if (status==‘e’)
            ArrS[sj].flag= 0;
   \}
5.2 for (every (ri,R,‘e’,pr) \in ip.datablock) \{
      ArrR[ri]= 0;
      for (every sj \in ArrR[ri].SObjects)
         if (ArrS[sj]=1)
            ArrR[ri].SObjects=
            ArrR[ri].SObjects \setminus \{sj\};
      else
         Oaxis= Oaxis \cup \{(ri,sj)\};
   \}

4. Conclusion
In this paper, we discussed the SB+-tree as a spatial indexing structure and how it can be used to process the Point-Containment and Regions-Selection operations. The reader can find more analytical and experimental results in [2].

References