Spatial Data Structures for Version Management of Engineering Drawings

Yasuaki Nakamura and Hiroyuki Dekihara

Department of Computer Science, Hiroshima City University
3-4-1, Ozuka-higashi, Asa-minami-ku,
Hiroshima, 731-3194
Japan

Abstract

In the engineering database system, multiple versions of a design including engineering drawings should be managed efficiently. The paper proposes an extended spatial data structure for efficient management of multiversion engineering drawings. The R-tree is adapted as a basic data structure. The efficient mechanism to manage the difference between drawings is introduced to the R-tree to eliminate redundant duplications and to reduce the amount of storage required for the data structure. The extended data structures of the R-tree, MVR and MVR* trees, are developed and the performances of these trees are evaluated. A series of simulation tests shows that, compared with the basic R-tree, the amounts of storage required for the MVR and MVR* trees are reduced to 50% and 30%, respectively. The search efficiencies of the R, MVR, and MVR* trees are almost the same.

Keywords: Spatial Data Structure, R-tree, Version Management, Design Database.

1. Introduction

Several version modeling techniques for the engineering database [1], and multi-version management structures for text data or non-spatial data [2-5] have been also proposed. However, the data structures for version management of drawings have not been proposed in the version modeling. Usually, a drawing contains a lot of spatial objects, such as points, line segments, shapes, and so on. The update process adds and deletes objects in a drawing. The drawing is then saved as a new version. Spatial data structures such as the R-tree [6] and the MD-tree [7] have been used in the CAD database to manage the drawings and spatial objects efficiently. However, data structures managing multiple versions of drawings have not been proposed. In the paper, the version management model of the engineering drawings is proposed. Our data structure, called the MVR-tree, is an extension of the R-tree that can manage the multiple versions of drawings. After evaluating the MVD-tree, we also propose an improved data structure, called the MVR*-tree.

In the following sections, a version model of engineering drawings and a spatial data structure, called the R-tree, are described in Section 2. In section 3, our data structures, the MVR and MVR* trees, are presented. A series of experimental results is shown in 4. As a result, it is shown that the MVR*-tree has much better performances than the MVR and the simple R-tree based method.

2. Spatial Data Structure and Version Management

Suppose design data X and Y have their own version histories and a version of X references some version of design data Y. Fig.1 shows an example version histories and references. Version 1 of Y, Y1, is referenced from versions 1 and 2 of X; X1 and X2. Y2 and Y3 are referenced from X3 and X3.1 respectively. Each version of design data has a drawing, for example, a draft of a part or circuit diagram. A new version is created by editing a portion of the drawing. In the design database, a hierarchical spatial data structure is usually used to manage the spatial objects in drawings. Especially, the R-tree [6] is one of the most popular data structure for spatial data management. In the section, at first, the algorithm of the R-tree is described briefly, because our proposed data structure is developed by extending the R-tree.

2.1 The R-tree [6]

In the R-tree, each node corresponds to a rectangular
region that encloses all regions of the lower nodes. The root corresponds to a region enclosing all data objects in the R-tree. Objects are only stored in leaf nodes. An internal node has at most M child nodes, and a leaf has at most M’ data. When an object is inserted into full leaf L, a new leaf is created and at least \((C \times M')\) objects in L are moved to the new leaf, where C is a constant. Usually, C is set to 0.4, because 0.4 gives better performances. The new leaf is added as a child of the L’s parent node. If the number of children of a node exceeds M, a new node is created and at least \((C \times M)\) children of the node are moved to the new node.

2.2 Version management of engineering drawings

A simple method to manage the multiple versions of a drawing is to create an R-tree corresponding to each version of the drawing, and to manage these R-trees in the database, as shown in Fig.2. The amount of storage required for these R-trees is proportional to the number of versions, when the number of objects in each version is the same. Generally, an update process modifies a small number of objects in a version or a small portion of the drawing in general, and a large part of the drawings and R-trees between versions are unchanged. Therefore, sharing the unchanged part between versions, the amount of the storage required can be reduced without the loss of search performances. Based on this idea, we expand the R-tree to share the unchanged objects and nodes with other R-trees.

3. Data Structure for Version Management: The MVR-tree

3.1 The MVR-tree

In a simple version management by R-trees, to create a new version of a design data, we create a copy of an old version, and then modify and save it as a new version. Even if we modify a small part of the design, the entire design data including unchanged data in the version are saved in the new version. To reduce the amount of storage, we propose a new data structure, called the MVR-tree, in which the unchanged objects and nodes are shared between several R-trees corresponding to versions. In the MVR-tree, we introduce the path copy method to manage the multiple version R-trees. The path copy method is described with referring Fig.3.

Assume new version V2 is created from version V1 of an R-tree. First, new root R2 corresponding to V2 is created by copying R1. R1 and R2 have the same child nodes as shown in Fig.3 (a). When object O1 is inserted into leaf L1 through node N1, all the nodes on the path from R2 to L1 are copied to the tree rooted with R2. N1 and L1 are copied as N1’ and L1’ in V2. O1 is then inserted in L1’, if L1’ is not full. An MVR-tree in Fig.3 (b) is obtained. When object O2 is inserted into full leaf L2 through R2, a copy of L2 is created. New leaf L2’ is then created and \((C \times M)\) objects in L2’1 are moved to L2’2. O2 is inserted into either L2’1 or L2’2. New node New1 is created to manage L2’1 and L2’2. New1 is added to R2 as a child. Fig.3 (c) illustrates this case.

If a new version is created from a version k-th tree, a new root is created by copying the root of the k-th version. In Fig.3 (d), the third version is created and an object is inserted in L2’1. The algorithm of creation of a new version and insertion of an object is presented in the followings.
(a) Construct a new version

CreateNewVersionFrom(Vk)
Vk: version of the R-tree.
/* create a new version by modifying version Vk. */
S1: Create new root Rn corresponding to a new version.
S2: Set the child pointers of Vk’s root to Rn. Return Rn.

(b) Insert an object

Insert(Vn, Ox)
Vn: R-tree version
Ox : Object
/* insert object Ox into version Vx */
S1: Let the root of Vn be R.
S2: Tracing from R, find leaf L in which Ox should be inserted.
S3: Copy the nodes and leaf L on the path from R to L that are not in Vn into Vn.
S4: Insert Ox into copied leaf L1 of L, if L1 is not full. Otherwise, create new leaf L2 and move (c x M’) objects in L1 to L2. Insert Ox into L1 or L2.
S5: If a new leaf is created, adjust the R-tree. This procedure is the same as that of the R-tree.

3.2 Improved MVR-tree: The MVR*-tree

The MVR-tree is efficient when the modifications are applied to a small area. However, even if the number of modified objects is small, the performances become worse when the modifications are applied to a wide area. To overcome the shortcoming of the MVR-tree, the MVR*-tree is developed by reducing the duplication of leaves in the MVR-tree. Namely, when an object is inserted into a leaf with a space, the object is inserted into the leaf without making a copy. In the case, since objects belonging to the different versions exit in a leaf, some token indicating active objects in the versions must be added to the data structure. This idea reduces the number of the duplications of leaves. Since the object occupancy rate in leaves is known to be 60~70% in the R-tree, the small number of modifications would not lead the large number of leaf splits if the modified area is wide.

The implementation of the idea is illustrated in Fig.4. We add a flag structure, called an active object identifier (AOI), between a lowest internal node and a leaf. AOI consists of an array of Boolean values and a pointer to a leaf. Boolean values indicate which objects in the leaf are active at the version. For examples, AOI 1 in Fig.4(a) has value 1 on the 1st and 2nd elements, and value 0 on the 3rd element. This means that the 1st and 2nd objects in a leaf are active at version V0 but the 3rd object is not active in versions V1 and V2. After inserting O1, as shown in Fig.4(b), all elements of AOI 1’ have value 1. AOI 1 and AOI 2 point the same leaf with 3 objects. However, the third object in the leaf is not active in version 1 because the 3rd element of AOI 1 is 0.

3.3 Construction of The MVR*-tree

The MVR*-tree is an improved MVR-tree with an array of active object identifier (AOI). AOI indicates which objects in a leaf are active at the version. This allows the multiple versions of R-trees in an MVR-tree can share a leaf. Therefore, in the MVR*-tree, a leaf is copied only when the leaf is full and an object is inserted in the leaf.

The algorithms of the MVR*-tree can be constructed by modifying the algorithm of the MVR-tree such that it makes AOI effective. The insertion algorithm of the MVR*-tree is presented as follows.

(a) Insert an object

InsertInMVR*(Vn, Ox)
Vn: R-tree version
Ox : Object
/* insert object Ox into version Vx */
S1: Let the root of Vn be R.
S2: Tracing from R to lower nodes, find AOI E pointing to leaf L in which Ox should be inserted.
S3: Copy the nodes on the path from R to E that are not in Vn into Vn.
S4: If E is not in Vn, make a copy of AOI E and set the pointer of the copied AOI to L. Set E to the copied AOI.
S5: If L is not full, insert Ox into L and set 1 to the corresponding element in E. Return.
S6: Otherwise, make copy L1 of L and copy E1 of E. Remove objects in L1 that are not active at this version.
S7: If L1 is full, create new leaf L2 and copy E1 of E, then move (C x M') objects in L1 to L2. Insert Ox by the R-tree’s algorithm. Adjust AOIs pointing to L1 and split leaf.

(b) Delete an object
In the deletion procedure, an object is not removed from a leaf immediately, but a corresponding element of AOI is set to be 0. An element with 0 means the corresponding object in a leaf is not active in a version.

Fig.4 shows an insertion and deletion processes of an MVR*-tree. At the first insertion of O1 in Fig.4(a), since leaf L is not full, object O1 is inserted in L1 through AOI 1. AOI 1 is a member of version 2. Therefore, a copy of AOI 1, AOI 1’, pointing to L1 is created and O1 is inserted in L1, as shown Fig.4(b). AOI 1 in V1 and AOI 1’ in V2 indicate which objects are active in each version of trees. When object O2 is inserted full leaf L2 in Fig.4(b), a copy of L2, L2’1, and a new leaf, L2’2, are created. Two objects in L2’1 are moved to L2’2, and O2 is then inserted. As a result, a MVR*-tree is Fig.4(c) is obtained. When object O3 in leaf L1 at version 2, O3 is not removed from L but the corresponding element (the first element) of AOI 1’ pointing to L1 in version 2 is set to be 0, as shown in Fig.4(d).

4. Experimental Results

To evaluate the performances and characteristics of the MVR-tree, the MVR*-tree, and the simple R-tree method, a series of simulation tests is carried out. In the simple R-tree method, R-trees managing the versions are created to manage multi-version drawings. An R-tree corresponding to a new version is created by copying an old version R-tree and modifying the contents. The roots of these R-trees are managed in the database. Hereinafter we represent this method the R-tree.

(1) Conditions of Experiments
The MVR and MVR* trees are expansions of the R-tree. The R-tree with the quadratic split algorithm is used as a basic structure in both cases of the MVR and MVR* trees. The number of maximum child nodes the R-tree is equal to 3, and the number of objects in a leaf is 20. The original version drawing, v0.0, with 100,000 objects is prepared. The entire data space is a 10,000x10,000 square area. Objects’ positions are generated randomly from a uniform distribution. Object are rectangles with the sides varying from 5 to 10 in length.

Versions, v1.0, v2.0, ..., and v6.0, are then created from version v0.0. Vv+1.0 is created from v0.0. 10,000 objects are added in each new version in any case. The following two cases are tested.

Case 1: 10,000 objects are inserted into a 100x100 square area at the center of the entire data space.
Case 2: 10,000 objects are inserted into the entire area randomly.

(2) Experimental Results
Table 1 shows the construction time of the R-tree, MVR-tree and MVR*-tree with 7 versions. The R-tree method requires more construction time. It takes almost same time to construct the MVR and MVR* trees.

The numbers of nodes and leaves in Case 1 experiment are shown in Fig.5 (a). The number of total leaves in the R-tree method is proportional to the number of versions. An R-tree with only one version contains approximately 5.5K nodes and the total nodes of the R-trees with 7 versions are approximately 48K. On the other hand, the number of total nodes in the MVR and MVR* trees are approximately 13.5K. The numbers of total leaves in the R, MVR, and MVR* trees are 62.6K, 17.9K, and 14.1K, respectively. Namely, the required storages for the MVR, and MVR* tree are reduced to 29% and 23% of the R-tree, respectively, when the objects are inserted in a small area.

The results of Case 2 experiment are shown in Fig.5 (b). Regarding the R-tree, the numbers of total nodes and leaves are almost equal to those of Case 1. However, the MVR-tree contains more nodes and leaves compared to Case 1. Especially, the number of leaves in the MVR-tree is 80% of the R-tree. In the MVR*-tree, the number of leaves...
is reduced to 1/3 of the MVR-tree. These results reveal that the MVR*-tree requires much less storage than the MVR-tree when the modification is applied in a wide area. The search performances of these methods are almost the same as shown in Table 2, because the structural properties of these methods are identical.

5. Conclusion

We proposed new data structures, the MVR-tree and MVR*-tree, for handling sets of spatial objects such as multiple versions of an engineering drawing, diagram, and so on. The experimental results reveal the better performances of the MVR and MVR* trees than the simple R-tree method when a modification is applied to a small portion of the data space. The number of nodes and leaves in the MVR-tree is 29% of the R-tree. When a modification affects a wide portion of the data space, however, the number of nodes and leaves in the MVR-tree decreases only 17% compared to the R-tree. To improve this, the MVR*-tree was developed. In the MVR*-tree, the number of nodes and leaves is 25% of the R-tree, in the case of the modification is local and is 53% in the case of the modification is applied wide area.

Since the engineering database is required to manage a large number of multi-version drawings efficiently, we plan to apply the MVR*-tree method as an index structure for engineering databases.

References


