Maintaining Connected Components in Quadtree-based Representation of Images

Vikrant Khanna, Phalguni Gupta, and C. J. Hwang

***Abstract*—In this paper, we have considered the problem of maintaining connected components in quadtree representation of binary images when a small portion of the image undergoes change. The batch approach to recalculate the connected components information is very expensive. Our algorithms update the quadtree and the connected components labeling when a homogeneous region in the quadtree is changed. Our algorithms visit less number of nodes as compared to the batch approach. For small changes in the image, the proposed algorithms save time to update the quadtree and take less time to update the labels of the components on the average.**

***Index Terms*—Connected components, data structures image processing, quadtree representation,**

I. INTRODUCTION

A QUADTREE is a hierarchical representation of an image and is based on recursive decomposition of the image in to homogeneous regions [1] where the criterion for homogeneity depends on the application. Initially, the whole image is treated as one region. If the region does not satisfy the homogeneity criteria, then it is divided in to four equal sub-regions which are termed as NW, NE, SW and SE quadrants and is shown in Fig. 1. The process of decomposition is repeated for each quadrant until the quadrant satisfies the homogeneity criteria. The advantage of using quadtree representation is that it allows focusing on subsets of data that are of interest without going through irrelevant data. It results in designing efficient image processing algorithms.

Finding connected components labeling [2] is a well-studied problem and there exist efficient algorithms to solve this problem in literature [3]-[6]. Eppstein [7] has given a lower bound of $O(\log n)$ to update this information per unit change i.e. one object pixel changing to background or vice-versa. In this paper we deal with the problem of maintaining the connected component labeling when only a small portion of the image is changing and images are represented using quadtree ([1], [8]).

As input, we normally have a set of p images $I = \{I_1, I_2, ..., I_p\}$ in which any pair of consecutive images I_j and I_{j+1} are very similar to each other and all the images have to undergo similar kind of processing. Traditionally, batch algorithms that process each image as separate entity have been employed. In order to find the connected components in each image, the standard batch approach is to find the connected components on each image separately. In contrast, dynamic approach is to maintain the connected component information of the previous image and update that information to represent the connected components in the new image.

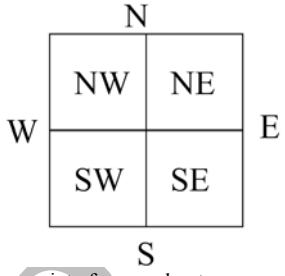

Fig. 1. Splitting an image into four quadrants.

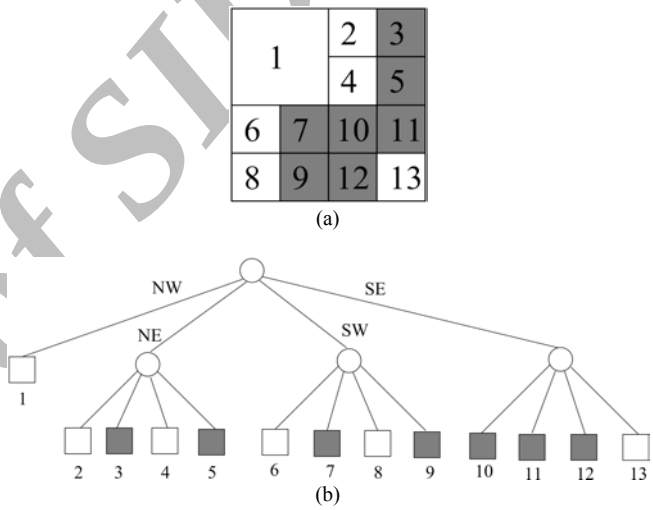

Fig. 2. (a) A binary image, (b) and its representation.

Thus, in the quadtree representation, there are three types of nodes namely `GRAY`, `OBJECT` and `BACKGROUND` ([1], [9]) which are depicted as circle, shaded square and white Square respectively. The `GRAY` node, the `OBJECT` node and the `BACKGROUND` node represent a region with both object as well as background pixels, a homogeneous region with only object pixels and that with only background pixels respectively. Fig. 2(a) shows a binary image and its corresponding quadtree is shown in Fig. 2(b). The homogeneous regions form the leaf nodes in the quadtree.

Assume, the number of `OBJECT` nodes in the quadtree is represented by δ_O while the number of `BACKGROUND` nodes in the quadtree is represented by δ_B. Suppose, a homogeneous region in the quadtree of the image I_0 with $\delta_O + \delta_B$ leaf nodes changes to yield image with I_1 with $\delta'_O + \delta'_B$ leaf nodes in the quadtree. Our objective is to

Manuscript received February 18, 2002; revised November 27, 2002. This work was supported in part by the Department of Space, India.

Vikrant Khanna is with Cadence India Ltd., India (e-mail: vkhanna@cadrence.com).

Phalguni Gupta is with the Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, Kanpur 208 016, India (e-mail: pg@iitk.ac.in).

C. J. Hwang is with the Department of Computer Science, Southwest Texas State University, Texas 78666 4616, USA (e-mail: ch01@swt.edu).

Publisher Item Identifier S 1682-0053(03)0139

design efficient algorithms to update the connected components labeling in the quadtree when the image undergoes such type of changes. Traditionally, the size of a component in a quadtree is obtained by the number of leaf nodes in that component [10].

In the following section, some operations on the quadtree that are used by our algorithms are discussed. Section III deals with various possible transformations that can be occurred in a homogeneous region. Algorithms to maintain the connected component labeling in each transformation are also presented in this section. In the next section, our algorithms are analyzed and compared with the batch algorithm. Concluding remarks are given in the last section.

II. Operations on Quadtrees

This section discusses some operations performed on quadtrees. These operations will be used in our algorithms.

A. Postorder Traversal Operation

Postorder traversal of the quadtree representation in which the leaf nodes will be visited as shown in Fig. 2, is equivalent to the Morton scan of the homogeneous regions in the binary image [8].

B. Neighborhood Operation

Given a node and a particular direction, this operation finds out all OBJECT nodes that represent neighboring regions of the node in the given direction. The algorithm that performs this operation is discussed in [10]. It is found to be an expensive operation and is extensively used in our algorithms to maintain the components labeling. So for completeness it becomes necessary to revisit the analysis of the complexity of the operation which is given in [10], [11].

Lemma 1: The average of the maximum number of nodes visited to find all the neighbors in a particular direction is at most five.

Proof: Given a node $\mathbf{S_R}$ at level i and a direction D, there exists a maximum of $2^{n-i}(2^{n-i}-1)$ neighbor pairs. $2^{n-i}\times 2^0$ nodes have their nearest common ancestor at level n, $2^{n-i}\times 2^1$ nodes at level $n-1$, …, and $2^{n-i}\times 2^{n-i-l}$ at level $i+1$. For each node at level i with a common ancestor at level j, the maximum number of nodes to be visited is $(j-i)+(j-i-1)+\sum_{k=0}^{i}2^k = 2(j-i-1)+2^{i+1}$. If we assume that the occurrence of a node **R** is equally likely at any level i and at any of the $2^{n-i}(2^{n-i}-1)$ positions at level i, then the average of the maximum number of nodes visited is

$$\frac{\sum_{i=0}^{n-1}\sum_{j=i+1}^{n}2^{n-i}.2^{n-j}.[2-(j-i-1)+2^{i+1}]}{\sum_{i=0}^{n-1}2^{n-i}.(2^{n-i}-1)} = \frac{\sum_{i=0}^{n-1}\sum_{j=0}^{n-1-i}2^{2n-2i-j-1}.(2j+2^{i+1})}{\sum_{i=0}^{n}2^{i}.(2^{i}-1)} \quad (1)$$

The numerator of (1) can be written as

$$\sum_{i=0}^{n-1}2^{2n-2i}\times\sum_{j=0}^{n-1-i}\frac{j}{2^j}+\sum_{i=0}^{n-1}2^{2n-2i}\times\sum_{j=0}^{n-1-i}\frac{1}{2^j}. \quad (2)$$

But

$$\sum_{i=0}^{n-1-j}\frac{j}{2^j}=2-\frac{n+1-i}{2^{n-1-i}} \text{ and } \sum_{i=0}^{n-1-j}\frac{1}{2^j}=2\times(1-\frac{1}{2^{n-i}}). \quad (3)$$

Substituting (3) in to (2) and using $\sum_{i=0}^{n-1}\frac{1}{2^{2i}}4^i=\frac{1}{3}(4-\frac{1}{2^{2n}})$, one can get the numerator as

$$\frac{20}{3}\times 2^{2n}-(3n+2)\times 2^{n+1}-\frac{8}{3}. \quad (4)$$

The denominator of (1) can be simplified as follows:

$$\frac{1}{3}\times(2^{2n+2}-3\times 2^{n+1}+2). \quad (5)$$

Substituting (4) and (5) in (1) one yields

$$\frac{\frac{20}{3}\times 2^{2n}-(3n+2)\times 2^{n+1}-\frac{8}{3}}{\frac{1}{3}\times(2^{2n+2}-3\times 2^{n+1}+2)}=5-\frac{3\times(3n+7)\times 2^{n+1}+18}{2^{2n+2}-3\times 2^{n+1}+2}$$

$$\approx 5 \text{ as } n \text{ gets large}$$

$$\leq 5$$

Hence the proof.

Instead of exploring all the neighbors separately, the common approach is to explore all the neighbors adjacent to a node in direction **D** simultaneously to reduce the number of nodes visited. We will be using the technique to find neighbors proposed in [11]. Assume that we have a function which takes a leaf node **R** and a direction **D** as input and finds a set of all neighboring OBJECT nodes of node **R** in direction **D**.

We will be using this operation in our dynamic algorithms for maintaining connected components. If we visit the neighbors by exploring adjacencies along a direction as is done in this function, rather than visiting each of them separately, then the following lemma holds.

Lemma 2: For any non-leaf node in the complete quadtree of depth n, on an average, there are at most $5n$ nodes in the subquadtree rooted at that node.

Proof: The number of nodes in the quadtree at level i is at most 4^i. Hence the total number of non-leaf nodes in the complete quadtree is

$$\sum_{i=0}^{n-1}4^i=\frac{4^i-1}{3}. \quad (6)$$

At each node at level i, there are $\sum_{j=0}^{n-i}4^j$ nodes in its subquadtree. This means that the total number of nodes in all the subquadtrees is given by

$$\sum_{i=0}^{n-1}4^i\sum_{j=0}^{n-i}4^j=\sum_{i=0}^{n-1}\frac{4^i(4^{n-i+1}-1)}{3}=\frac{n\times 4^{n+1}}{3}-\frac{4^n+1}{9}. \quad (7)$$

Hence, from (6) and (7), the average number of nodes in a subquadtree rooted at a non-leaf node is given by

$$\frac{3n\times 4^{n+1}-(4^n-1)}{3\times(4^n-1)}=\frac{12n\times(4^n-1)-(4^n-1)+12n}{3\times(4^n-1)}=$$
$$=4n-\frac{1}{3}+\frac{4^n}{4^n-1}$$
$$<5n \quad \text{as } n \text{ gets large}$$

The neighbor finding operation is mainly required to define the *interleaved breadth first search* (IBFS) operation which is discussed below.

C. Merging Operation

During modification in the original image, a GRAY region in the quadtree representation may become a homogeneous region. This operation updates the quadtree by merging the leaves and their parent to form one node representing a homogeneous region. Suppose the depth of the quadtree is n. Then,

Lemma 3: The merging operation requires visiting at most 5n nodes to update the quadtree.

D. Building Subquadtree Operation

Notice that any modification in a homogeneous region may cause the conversion of the region into a GRAY region and hence, it needs to update the quadtree. This operation uses the algorithm given in [12] and the number of nodes visited by this algorithm is bounded by the number of leaf nodes, δ_N, in the subquadtree. Hence,

Lemma 4: The building subquadtree operation visits at most $O(\delta_N)$ nodes.

E. Interleaved Breadth First Search (IBFS)

In standard breadth first search (BFS) on quadtrees, we are given a node to begin with. Using that node as seed, we traverse OBJECT nodes in breadth first order i.e. visit all the immediate neighbors of that node in all directions before visiting the neighbor's neighbors. Note that the workset in the standard BFS is implemented as queue. In the interleaved breadth first search (IBFS), we have a pair of nodes which serve as seed for two separate BFS processes respectively. The two BFS processes visit the nodes starting from the seeds, in an interleaved fashion. This technique is discussed in [13]. In this section we present a modification of the technique to suit our purpose.

Suppose for region **R**, we get a set $\mathbf{S_R}$ of neighboring nodes in all directions. Out of the nodes in $\mathbf{S_R}$, our objective is to find those that are connected and those that are disconnected. For each component except the largest one, we also desire to build a list which contains all the nodes in that component. Note that the largest component is the one having the largest number of nodes. We start with a pair of nodes from $\mathbf{S_R}$. Let us assume that we have a function GET_PAIR() that returns a different pair of node from the set, every time it is called. For each node we put it in its corresponding workset if it has not been visited and start a separate BFS process for each node. The first BFS process visits a node after removing it from the first workset. It puts OBJECT neighbors of the visited node in the corresponding workset. Then it inserts the visited node in the list corresponding to the component to which the node should belong and stops. Now the other BFS process starts and visits a node after removing it from the second workset. It puts all OBJECT neighbors of the visited node in the corresponding workset. Then it inserts the visited node in the list corresponding to that component to which the node should belong and stops. Again the first BFS process starts and does its job. This goes on and the processes visit one node at a time.

As soon as the workset for a particular process becomes empty, it means that the particular process has visited all the nodes in the corresponding component and execution of IBFS is stopped. The component whose workset has become empty is the smaller component since it constituted of less number of nodes. The list corresponding to that component has all the nodes of that component.

It may so happen that during the IBFS, one BFS process may visit a node that has been visited earlier by the other BFS process. This means that the nodes visited by the two BFS process belong to the same component. In such a case, we stop the IBFS, merge the two lists and worksets. Moreover, if we visit a node that is in the set $\mathbf{S_R}$, we remove that node from the set so that it does not appear in the next pair of nodes. When we select a pair of nodes from the set $\mathbf{S_R}$ before starting IBFS, it may so happen that a node may belong to a component that has been partially visited by an earlier IBFS. For such a node, we must be having a corresponding list constructed earlier. In this case, we compare the size of the lists associated with the nodes in the pair and run a BFS process on the node with smaller list. This process continues till the size of both the lists become equal and all the nodes in the component are visited. After this, if still the two components appear to be separate and not all the nodes in each component have been visited, then we start the IBFS with the existing worksets and lists.

III. Homogeneous Region Transformations

In this section, we discuss various transformations that a homogeneous region can undergo when the image changes in that region. We also present algorithms to update the quadtree as well as the connected component labeling for these transformations.

Effectively, all the changes can be grouped into those affecting a single region and then we can use our algorithms to handle the changes in every affected region. We assume that the changed pixels have already been grouped and we have identified the affected regions before we maintain the labeling of connected components in our quadtree. The region transformations can be of four types:

- *BTO Transformation:* In this transformation, all the BACKGROUND pixels in the homogeneous region change to OBJECT pixels. This means that the BACKGROUND node will have to change its Nodetype to OBJECT to reflect this change, apart from updating the pixel information stored in the node.

- *OTB Transformation:* This transformation is exactly the opposite of BTO transformation. Here, all OBJECT pixels in the homogeneous region change to BACKGROUND pixels. This means that the corresponding node in the quadtree will have to change its Nodetype to BACKGROUND.

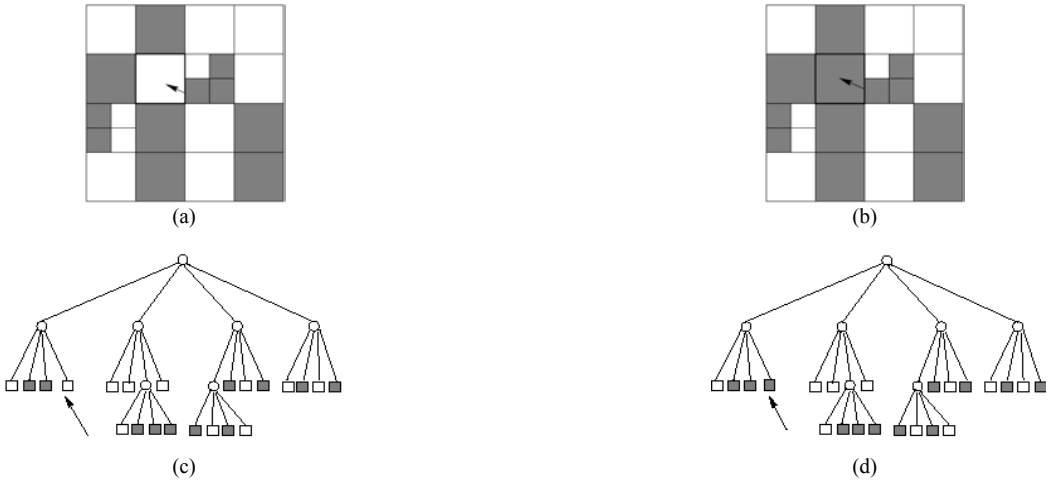

Fig. 3. BTO transformation: (a) original image, (b) BTO transformed image, (c) quadtree original image, (d) quadtree of BTO transformed image.

• *BTG Transformation:* In this transformation, some BACKGROUND pixels in a homogeneous region change to OBJECT pixels. This makes the region non-homogeneous and has to be decomposed further till we get homogeneous regions. In the quadtree, it means that the corresponding BACKGROUND node is transformed into a gray node. Apart from updating the BACKGROUND node to the GRAY node, we have to build the subquadtree with the affected node as the root. This could be easily done using the algorithm suggested by Shaffer et al. [12] and update the quadtree by modifying the pointers.

• *OTG Transformation:* In this transformation, some OBJECT pixels in a homogeneous region change to BACKGROUND pixels. As was the case in BTG, we have to decompose the region further. In the quadtree, this requires changing the OBJECT node to the GRAY node and builds the subquadtree with the affected node as the root. The subquadtree can be built in the same way as done in BTG transformation.

Before we present the details of each transformation, let us define some symbols used in our analysis. δ_O and δ_B are the number of OBJECT and BACKGROUND nodes in the quadtree representation of the image before modification and let n be the quadtree's depth. Similarly, δ'_O and δ'_B are the number of OBJECT and BACKGROUND nodes in the quadtree representation of the image after modification and let n' be the quadtree's depth. Suppose a leaf node in the quadtree of the original image is replaced by a subquadtree because of some modification in the original image. Then δ_N is the number of leaf nodes in that subquadtree.

A. BACKGROUND to OBJECT Node Transformation

Fig. 3 shows the *BTO* transformation. Fig. 3(a) is the original image in which the heavily marked region undergoes the *BTO* transformation to give us the image in Fig. 3(b). The quadtree representation of the original image and the modified image is given by Figs. 3(c) and 3(d), respectively.

Let the affected region be **R** and the neighboring regions (and corresponding nodes in the quadtree) lie in **p** different components. This means that the neighbors are labeled with **p** different labels. Once the region changes and the node changes its Nodetype from BACKGROUND to OBJECT, all the neighbors will become connected. If they belong to the same component i.e. **p** is 1, then there is nothing to do. But if some neighbors belong to different components, we need to relabel nodes belonging to **p-1** components to denote this change. Here we apply the smaller component relabeling heuristic suggested by Even et al. [13]. This heuristic says that the label of the component having largest number of nodes is used to relabel the nodes belonging to other components. Finding the component with largest number of nodes is easy since we keep the count of the nodes associated with each of the labels. So, we relabel the nodes belonging to smaller components with the label of the largest component. Size of a component is the number of nodes in it. The detail of the algorithm to handle the *BTO* transformation is given below.

BTO Algorithm

1. Update the Quadtree. Let the affected region be **R**.
2. Find $\mathbf{S_R}$, OBJECT neighbors of **R** in all four directions.
3. Find L_R, the set of labels assigned to nodes in $\mathbf{S_R}$.
4. Keep only one node per label in $\mathbf{S_R}$ and remove all other nodes in $\mathbf{S_R}$.
5. Among the labels in $\mathbf{L_R}$, identify the label l of the largest component.
6. Remove the node in $\mathbf{S_R}$ corresponding to that label.
7. Assign the label l to region **R** and update the count
8. **forall** $\alpha \in \mathbf{S_R}$ **do**
9. $W_\alpha = \{\alpha\}$
   while $W_\alpha \neq \phi$ **do**
10. Remove a member x from W_α
11. Relabel x with l and update the count
12. Find $\mathbf{S}_x$, the set of OBJECT neighbors of x in all directions
13. **forall** $y \in \mathbf{S}_x$ which do not have the label l **do**
14. Put y in W_α
    done

The most expensive operation in this and following algorithms is following pointers to visit nodes. So, we present the analysis in terms of the nodes visited by the algorithms.

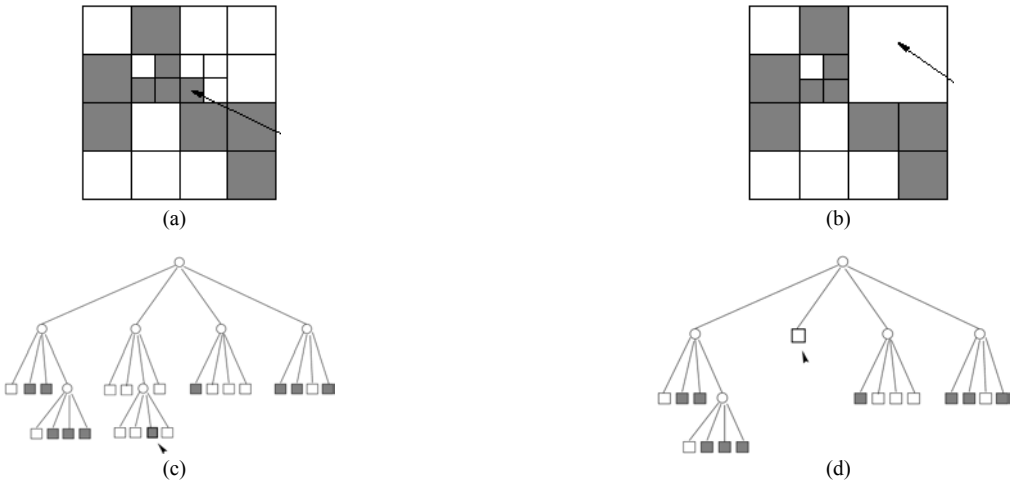

Fig. 4. OTB transformation: (a) original image, (b) OTB transformed image, (c) quadtree original image, (d) quadtree of OTB transformed image.

Lemma 5: The total number of nodes visited by the BTO algorithm is $O(\delta_O + n)$.

Proof: When a BACKGROUND node transforms to OBJECT node, it may cause merging of sibling nodes. Merging nodes at a level means visiting the father node and the sibling nodes and replacing the father node by an OBJECT leaf node. The merging may not stop here and in worst case, we may have to perform merging at each level. This means that step 1 will require visiting at most $5n$ nodes. By Lemma 1, step 2 will visit at most 20 nodes on an average. To relabel the nodes belonging to smaller components, we have to explore the adjacency of each node belonging to those components in for loop (steps 9-14). Now the sum of the sizes of the smaller components is bounded by the number of OBJECT nodes denoted by δ_O. So the total number of nodes visited in for loop (steps 9-14) is at most $20\,\delta_O$ on an average. Hence the number of nodes visited is bounded by $O(\delta_O + n)$.

B. OBJECT to BACKGROUND Node Transformation

The *OTB* transformation is demonstrated with the help of Fig. 4. The original image and the modified image are shown in Figs. 4(a) and 4(b), respectively. Corresponding quadtree representations are given in Figs. 4(c) and 4(d) respectively. Transformation of a node from OBJECT type to BACKGROUND type may cause merging of nodes as shown in the figure.

The algorithm given below updates component labeling under *OTB* transformation. When a region **R** is an OBJECT region, all its neighboring OBJECT regions are connected and belong to the same component. This means that all OBJECT neighbors of an OBJECT node have the same label as that node. Once that node becomes a BACKGROUND node, the neighboring nodes may become disconnected and hence form separate components. Let OBJECT neighbors of **R** be grouped into set $\mathbf{S_R}$.

OTB Algorithm:

Let the original region be **R** and its label be l.

1. Find **R**, OBJECT neighbors of **R** in all four directions.
2. Update the Quadtree.
3. $\{n_1, n_2\}$=GET_PAIR($\mathbf{S_R}$).
   while n_1 and n_2 are not NULL **do**
4. IBFS(n_1, n_2).
5. $\{n_1, n_2\}$=GET_PAIR($\mathbf{S_R}$).
   done
6. Identify the largest list of nodes.
   It corresponds to the largest component created out of the original component
   forall smaller lists **do**
7. Get a new label. Label all the nodes in that list with the new label
   done

To start with, we take a pair of neighboring node and do an IBFS on that pair. If the pair is still connected, we will encounter a common node and hence we do not require any relabling for the region to which the pair belongs. If during the course of IBFS, we encounter an OBJECT node in the neighborhood of **R**, then it means that node too is in the same region. So we remove that node from the set $\mathbf{S_R}$. If the nodes in the pair are not connected anymore, then a stage will come when we have scanned all the nodes in the smaller component and IBFS stops. A complete list of nodes is formed for the smaller component. A stage will come when we have exhausted all pairs of neighbors. At this point, the largest list corresponds to the largest component. All other lists represent smaller components. Here we apply the smaller component relabling heuristic to relabel the smaller components. For each smaller list, we get a new label and relabel all the nodes in the list by that label.

Lemma 6: The total number of nodes visited by OTB algorithm is $O(\delta_O + n)$.

Proof: The algorithm visits at most 20 nodes in step 1. Merging in step 2 may cause it to visit at most $5n$ nodes. GET_PAIR() function simply returns any two nodes from the set $\mathbf{S_R}$ and this pair cannot be repeated again because one of the nodes in the pair gets deleted from the set before another call to GET PAIR() function. So finding node pairs is a constant time operation requiring no node visits. The number of OBJECT nodes visited by the all the IBFS invoked in step 4 within the loop is bounded by O and we explore the adjacencies of the visited OBJECT nodes in all 4 directions. Therefore by Lemma 1, we visit at most $20\,\delta_O$ nodes. The number of nodes visited in step 7 is bounded by δ_O. The number of nodes visited by the algorithm is hence bounded by $(\delta_O + n)$.

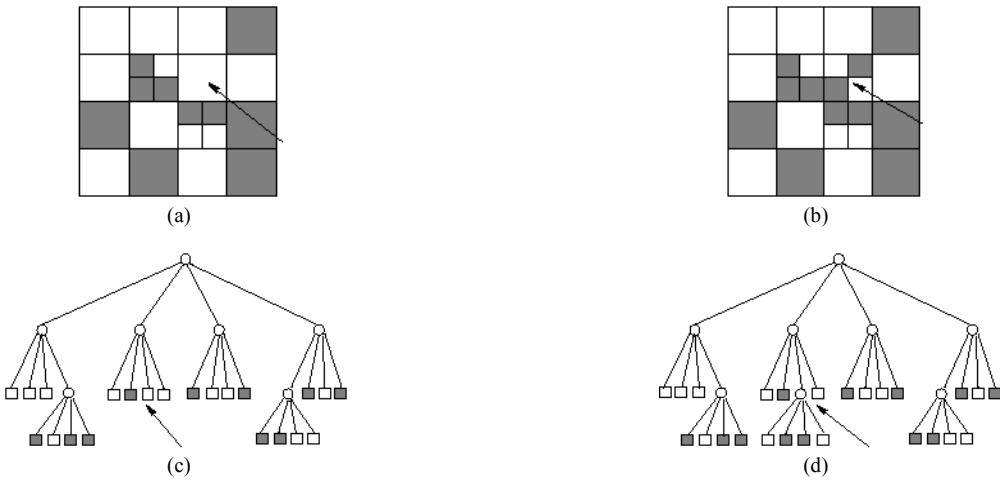

Fig. 5. BTG transformation: (a) original image, (b) BTG transformed image, (c) quadtree original image, (d) quadtree of BTG transformed image.

C. BACKGROUND to GRAY Node Transformation

The *BTG* transformation is shown with the help of Fig. 5. The original image and the modified image are given in Figs. 5(a) and 5(b), respectively; while their quadtree representations are in Figs. 5(c) and 5(d), respectively. When a node changes to the GRAY node, we have to scan the region and construct a subquadtree. Then we replace the leaf node representing that region by the subquadtree obtained. In order to construct the subquadtree, we can use the algorithm by Shaffer et al. [12].

The algorithm for *BTG* transformation is presented below. We first find the neighbors of the affected region and then update the quadtree to reflect the changes in that region. We label the newly created OBJECT nodes that are adjacent to a node in $\mathbf{S_R}$ and merge equivalence labels in the equivalence table. This will make all the boundary nodes in the modified region be labeled with the existing labels. Once the boundary nodes in the modified region are labeled, we label the rest of the newly created nodes by the classical component labeling algorithm by Samet [10]. After this is done, the components that become connected will have their labels in the same equivalence class. So, for each equivalence class, we identify the label, say l, used for largest number of nodes. Then we pick a node from $\mathbf{S_R}$ with a label in the same equivalence class as l and do a BFS starting from that node. We replace the labels of the nodes encountered in the BFS with the label l. This step is repeated for each equivalence class of labels.

Algorithm: BTG

Let $\mathbf{R}$ be the affected region.

1. Find $\mathbf{S_R}$, the set of OBJECT neighbors of $\mathbf{R}$ in all directions
2. Initialize the Equivalence Table with the labels of $\mathbf{S_R}$
3. Update the Quadtree
   forall α in $\mathbf{S_R}$ **do**
4. Find $\mathbf{S}_\alpha$, neighbors of α in the affected region.
   forall $x \in \mathbf{S}_\alpha$ **do**
5. **if** x is unlabeled then label it with α' s label
6. **if** x' s label is different **then**
   merge the equivalence classes of the two labels
7. Label the updated portion of the quadtree using classical method
8. Update the counts of nodes associated with each label in the Quadtree
9. For each equivalence class, identify the label used for maximum number of nodes
10. Remove the nodes from $\mathbf{S_R}$ with those labels
    forall $\alpha \in \mathbf{S_R}$ do
11. Let l be the label of α
12. Let the label identified in Step 9 for the equivalence class of l be m
13. $W_\alpha = \{\alpha\}$
    while $W_\alpha \neq \phi$ **do**
14. Remove a member x from W_α and relabel x with m
15. **if** x is in $\mathbf{S_R}$ then remove it from $\mathbf{S_R}$
16. Find $\mathbf{S}_x$, the set of OBJECT neighbors of x in all directions
    forall $y \in \mathbf{S}_x$ which have the label l do
17. Put y in W_α

done

Lemma 7: The total number of nodes visited by the BTG algorithm is bounded by $O(\delta_O + \delta_N + n)$.

Proof: At most 20 nodes are visited in finding the set $\mathbf{S_R}$ in step 1. Updating the quadtree in step 3 has the time complexity bounded by the number of nodes created which is at most $5n$ on an average by Lemma 2. Suppose the number of leaf nodes in subquadtree is δ_N. Then the classical algorithm for labeling the regions in the subquadtree will visit at most $O(\delta_N)$ nodes [10]. For steps 11-17, the number of nodes that are relabeled is bounded by δ_O and for each such node we explore the adjacency in all four directions. So the number of nodes being visited in these steps is at most $20\,\delta_O$. Hence the lemma holds.

D. OBJECT to GRAY Node Transformation

The *OTG* transformation is demonstrated with the help of Fig. 6. The original image is presented in Fig. 6(a) and its corresponding quadtree is presented in Fig. 6(c). Similarly, the modified image is presented in Fig. 6(b) and its respective quadtree is presented in Fig. 6(d). Similar to the *BTG* transformation, we have to construct a subquadtree for the affected region and replace the leaf node with the obtained subquadtree. The neighbors of the affected node $\mathbf{R}$ are found out and put in set $\mathbf{S_R}$. Then the

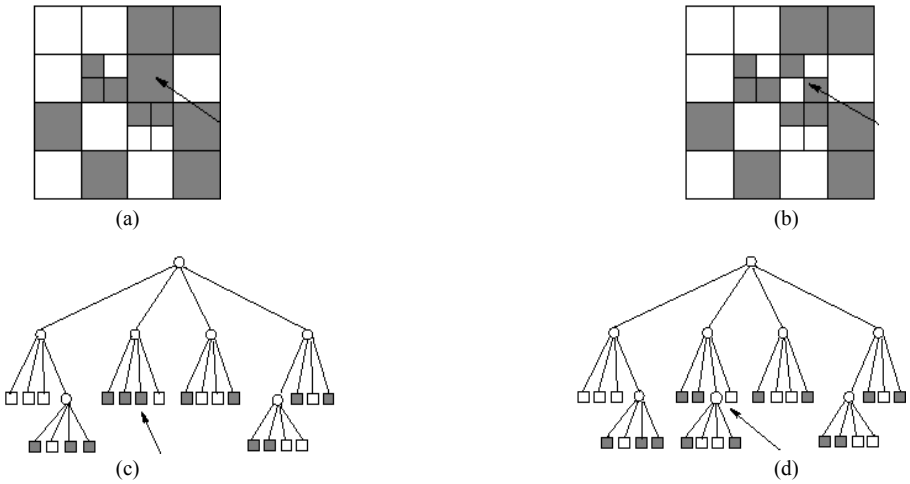

Fig. 6. OTG transformation: (a) original image, (b) OTG transformed image, (c) quadtree original image, (d) quadtree of OTG transformed image.

quadtree is updated. We label all the newly created nodes in the neighborhood of the nodes in $\mathbf{S_R}$, with the label of **R.** This is followed by labeling the subquadtree using the classical method. Now, we have to identify all the disconnected components that were previously connected. This is done by applying the IBFS on the pair of nodes obtained from the set $\mathbf{S_R}$. After this process is complete, we are left with lists of nodes corresponding to different components created. We identify the largest list. For all other lists, we label all the nodes in the list with a new label. The details of the OTG algorithm are presented below.

OTG Algorithm:

Let **R** be the affected region and its label be l

1. Find $\mathbf{S_R}$, the set of OBJECT neighbors of **R** in all directions
2. Update the Quadtree.
3. **forall** α in $\mathbf{S_R}$ **do** Label the neighbors of α in the updated portion with label l
4. Label the updated portion of the quadtree using classical method.
5. Update the counts of nodes associated with the labels
6. $\{n_1, n_2\}$=GET_PAIR($\mathbf{S_R}$).
   while n_1 and n_2 are not NULL **do**
7. IBFS(n_1, n_2).
8. $\{n_1, n_2\}$=GET_PAIR($\mathbf{S_R}$).
9. Identify the largest list of nodes.
   It corresponds to the largest component created out of the original component
   forall smaller lists **do**
10. Get a new label. Label all the nodes in that list with the new label
    done

Lemma 8: The total number of nodes visited by the OTG algorithm is bounded by $O(\delta_O + \delta_N + n)$.

Proof: Finding all the members of the set $\mathbf{S_R}$ in step 1 requires visiting at most 20 nodes by Lemma 1. Updating the quadtree will require visiting at most 5*n* node on an average by Lemma 2. The number of nodes visited in step 3 is bounded by $O(\delta_O)$. Step 4 will require at most $O(\delta_N)$ nodes where the subquadtree has δ_N leaf nodes. The number of nodes visited by all the IBFS within the loop in step 7 are bounded by $20\,\delta_O$. The number of nodes visited in step 10 is bounded by δ_O. Hence the number of nodes visited by the OTG algorithm is bounded by $O(\delta_O + \delta_N + n)$.

IV. Comparison with Batch Approach

Suppose we have an image set I in which consecutive images I_j and I_{j+1} are very similar to each other. In the batch approach for maintaining connected components, we first build the quadtree and then label the components. This procedure is done for every image in the set. Let us just concentrate on a pair of images I_j and I_{j+1}. In order to build the quadtree of the image I_j, we have to scan the whole image and using the algorithm suggested by Shaffer et al. [12] in time bounded by $O(m)$ where m is the number of nodes in the quadtree representation of that image. Let the quadtree has $\delta_O + \delta_B$ leaf nodes. Then the time required to construct the quadtree is bounded by $O(\delta_O + \delta_B)$ since the number of nodes in a quadtree is bounded by the number of leaf nodes it has.

To find connected components in the quadtree, the time complexity of the algorithm by Samet [10] is $O(\delta_O + \delta_B)$ if we assume that we have a linear cost maintenance of equivalence table as suggested in [4]. This requires postorder traversal of the whole quadtree twice. We have to repeat the same proceedure for the next image I_{j+1}. Creation of the quadtree of the modified image I_{j+1} is bounded by $O(\delta'_O + \delta'_B)$ if the quadtree has $\delta'_O + \delta'_B$ leaf nodes. Then finding connected components in this quadtree takes $O(\delta'_O + \delta'_B)$ time. Hence the time complexity of the batch approach to update labels when a homogeneous region undergoes change is $O(\delta_O + \delta_B + \delta'_O + \delta'_B)$.

In contrast, our approach for maintaining connected components requires constructing the quadtree just once for the image set I. We construct the quadtree for the first image and for successive images; we update the quadtree of the previous image. For this, we assume that instead of providing the whole images, we are just given the differences between an image and its predecessor image. So, we just have to update those portions of the quadtree that are undergoing changes, rather than building the whole quadtree from the scratch. Hence if the difference between consecutive images is not much, then only a minor portion of the quadtree need to be updated. The updation cost per

unit change is bounded by $O(\delta_N)$ where N is the number of leaf nodes in the updated portion of the quadtree.

Theorem 1: Algorithms to maintain connected components on quadtrees visit at most $O(\delta_O + \delta_N + n)$ nodes.

Proof: For a unit change, a region may undergo any one of the four transformations discussed in the previous section. The time complexity of our algorithms for each of these transformations is bounded by $O(\delta_O + \delta_N + n)$ nodes, by Lemmas 5, 6, 7 and 8. Hence the theorem holds.

Since the subquadtree with δ_N leaf nodes is a part of the quadtree of the modified image, $\delta_N \le (\delta'_O + \delta'_B)$. Moreover, for large images, n is very small compared to $\delta_O + \delta_B$. This means that the number of nodes visited by our algorithm has a bound lower than that of the batch method. If the number of changes involved is very few i.e. the consecutive images are very similar, our algorithms will not only save time in updating the quadtree, but also update the labels faster than the classical algorithm on an average. Theoretically, in the worst case, the updation of labels in our algorithms for maintaining connected components can be as bad as the classical algorithm employed in the batch approach.

V. CONCLUSIONS

We have presented dynamic algorithms to update the connected components labeling in quadtree representation of binary images, when a homogeneous region undergoes some changes. Our algorithms save on the rebuilding cost of the quadtree for every change. The bound on the number of nodes visited by our algorithms can be reduced further by maintaining more history information [14]. It is shown that for small changes in an image represented by a quadtree, on an average, algorithms presented in this paper take less amount of time to update the quadtree and also to label connected components as compared to the classical algorithm. However, in the worst case, our algorithms are as bad as the classical algorithm. It is worth to determine the threshold value experimentally, for which the proposed algorithms are found better than the classical algorithm.

ACKNOWLEDGEMENT

Authors are thankful to the anonymous referees for their valuable comments.

REFERENCES

[1] H. Samet, "The quadtree and related hierarchical data structures," *Computing Surveys*, vol. 16, no. 2, pp. 187-260, Jun. 1984.

[2] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, Addison Wesley, 1998.

[3] G. A. Cheston, "On-line connectivity algorithms," *Networks*, vol. 14, no. 1, pp. 83-94, Spring 1984.

[4] M. B. Dillencourt, H. Samet, and M. Tamminen, "A general approach to connected components Labeling for arbitrary image representations," *J. ACM*, vol. 39, no. 2, pp. 253-280, Apr. 1992.

[5] R. Lumia, "A new three dimensional connected components algorithm," *Computer Vision, Graphics and Image Processing,* vol. 23, no.2, pp. 207-217, Aug. 1983.

[6] O. Zuniga, R. Lumia, and L. Shapiro, "A new connected components algorithm for virtual memory computers," *Computer Vision, Graphics and Image Processing*, vol. 22, no. 2, pp. 287-300, May 1983.

[7] D. Eppstein, "Dynamic connectivity in digital Images," *Information Processing Letters,* vol. 62, no. 3, pp. 121-126, May 1997.

[8] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, 1990.

[9] H. Samet, "Region representation: quadtrees from binary arrays," *Computer Graphics and Image Processing*, vol. 13, no. 1, pp. 88-93, May 1980.

[10] H. Samet, "Connected component labeling using quadtrees," *J. ACM*, vol. 28, no. 3, pp. 487-501, Jul. 1981.

[11] H. Samet, "Neighbor finding techniques for images represented by quadtrees," *Computer Graphics and Image Processing*, vol. 18, no. 1, pp. 37-57, Jan. 1982.

[12] C. A. Shaffer and H. Samet, "Optimal quadtree construction algorithms," *Computer Vision, Graphics and Image Processing*, vol. 37, no. 3, pp. 402-419, Mar. 1987.

[13] S. Even and Y. Shiloach, "An on-line edge deletion problem," *J. ACM*, vol. 28, no. 1, pp. 1-4, Jan. 1981.

[14] G. Ramalingam, *Bounded Incremental Computation*, Springer-Verlag, 1996.

Vikrant Khanna recieved the B. Tech. degree in Computer Science and Engineering from M.N.R.E.C at Allahabad, India in 1998 and M. Tech. degree from Indian Institute of Technology Kanpur, India in 2000. His M. Tech. thesis was entitled "New Algorithms For Some Image Processing Problems In Remote Sensing Applications" and was supported by ISRO, India. At present he works for Cadence Design Systems at Noida, India. His research interests include genetic algorithms, on-line algorithms and physical design layout problems.

Phalguni Gupta received the Doctoral degree from Indian Institute of Technology Kharagpur, India in 1986. He works in the field of data structures, sequential algorithms, parallel algorithms, on-line algorithms. From 1983 to 1987, he was in the Image Processing and Data Product Group of the Space Applications Centre (ISRO), Ahmedabad, India and was responsible for software for correcting image data received from Indian Remote Sensing Satellite. In 1987, he joined the Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, India. Currently he is an Associate Professor at the department. He is responsible for several research projects in the area of image processing, graph theory and network flow. Dr. Gupta is a member of the Association Computing Machinery (ACM).

C. J. Hwang holds a Doctoral Degree in Mathematics from Louisiana State University and a Masters of Science in Mathematics/Computing Science from National Taiwan University. Dr. Hwang has also received certificate from Harvard Executive Leadership Program seminars. He began his teaching career over 22 years ago at Purdue University and was an Associate Professor. Dr. Hwang has also taught at University of Maryland as visiting faculty and currently teaches at Southwest Texas State University as a Professor of Computer Science. Some of Dr. Hwang's courses include object-oriented analysis and design, Research in object-oriented systems development, database systems modeling development, and algorithms. Dr. Hwang has also worked as a Senior Consultant at some international IT consulting companies for over 15 years. He has played a major role in numerous IT projects including multimedia, e-commerce and image compression projects. His expertise and research interest covers a wide range of topics including data modeling and design, object oriented design modeling, e-commerce and image processing algorithms.