*Journal of Visual Languages and Computing* (1999) 10, 147–164
Article No. jvlc.1998.0107, available online at http://www.idealibrary.com on IDEAL

# 2D C-Tree Spatial Representation for Iconic Image

FANG-JUNG HSU,*† SUH-YIN LEE† AND BAO-SHUH LIN‡

†*Institute of Computer Science and Information Engineering, National Chiao Tung University, 1001 Ta Hsueh Road, HsinChu, Taiwan 30050, People's Republic of China, e-mail: fjhsu@info4.csie.nctu.edu.tw*
‡*Computer & Communication Research Laboratories, Industrial Technology Research Institute, HsinChu, Taiwan*

*Accepted 1 October 1998*

The 2D string approaches provide a natural way of constructing iconic indexing for images. The 2D C-string representation with an efficient cutting mechanism is more characteristic of spatial knowledge and efficient in the representation of iconic images. However, the computation of object ranks in a 2D C-string might make the inference of spatial reasoning somewhat complicated. This shortcoming is overcome by the 2D C-tree representation. The 2D C-tree not only keeps the comprehensive spatial knowledge in the original 2D C-string, but also the ordered labeled tree is more suitable for spatial reasoning and image retrieval. The spatial knowledge can be derived directly from the inference rules embedded in the characteristic structure of the 2D C-tree representation.


## Notation

| | |
|---|---|
| $X_i$ | the $i$th subpart of symbolic object $X$ |
| $A_b$ | the begin-bound of object $A$ |
| $A_e$ | the end-bound of object $A$ |
| $R$ | the root of a 2D C-tree |
| $s_i$ | the $i$th immediate descendant of node $S$ |
| $\varepsilon$ | empty-node |
| $\{C, D\}$ | a set-node containing objects $C$ and $D$ |
| $\mathfrak{I}$ | a rooted tree |
| $\alpha$, $\beta$ | the Dewey decimal notation (D-notation) of node |
| $\alpha/\mathfrak{I}$ | a subtree rooted at $\alpha$ in $\mathfrak{I}$ |
| $\lvert\mathfrak{I}\rvert$ | the size of tree $\mathfrak{I}$ |
| $S_{\mathfrak{I}}(\alpha)$ | the label of $\alpha$ in $\mathfrak{I}$ |
| $D_{\mathfrak{I}}(\alpha)$ | the depth of $\alpha$ in $\mathfrak{I}$ |
| $L_{\mathfrak{I}}(\alpha)$ | the level of a subtree rooted at $\alpha$ in $\mathfrak{I}$ |
| $P_{\mathfrak{I}}(\alpha)$ | the set of predecessors of $\alpha$ in $\mathfrak{I}$ |
| $P^i_{\mathfrak{I}}(\alpha)$ | the $i$th predecessor of $\alpha$ in $\mathfrak{I}$ |

*Corresponding author.

1045-926X/99/020147+18 $30.00/0 

$C_{\mathfrak{I}}(\alpha)$ the number of immediate descendants of $\alpha$ in $\mathfrak{I}$
$p_{\mathfrak{I}}(\alpha)$ the set of legitimate descendants of $\alpha$ in $\mathfrak{I}$
$p^i_{\mathfrak{I}}(\alpha)$ the $i$th legitimate descendant of $\alpha$ in $\mathfrak{I}$
$q_{\mathfrak{I}}(\alpha)$ the set of lateral descendants of $\alpha$ in $\mathfrak{I}$
$q^i_{\mathfrak{I}}(\alpha)$ the $i$th lateral descendant of $\alpha$ in $\mathfrak{I}$
$f_{\mathfrak{I}}(\alpha)$ the postfix ordering of $\alpha$ in $\mathfrak{I}$
$\theta_{\mathfrak{I}}(\alpha)$ the leading descendant of subtree $\alpha/\mathfrak{I}$

## 1. Introduction

IN IMAGE INFORMATION SYSTEMS, one of the most important methods for discriminating the images is the perception of objects and the spatial relationships that exist among them in the desired images. Therefore, how images are stored and the capability of assembling queries on objects and their spatial relationships in a database are important design issues of image database systems. The data structure, called 2D string [1], to represent the spatial knowledge in symbolic pictures was introduced by Chang *et al.* This approach allows a natural way of constructing iconic indexes for images. In order to describe more accurately the spatial relations for images of arbitrary complexity, many extended representations were proposed. 2D G-strings [2, 3] extended the idea of symbolic projection [4] by introducing the cutting mechanism to describe images with overlapping objects. The cuttings are performed at all extreme points of symbolic objects in the image viewing from *x*- and *y*-projection, respectively. A more efficient cutting mechanism was introduced in the 2D C-string representation [5]. Basically, the cutting of 2D C-string is performed only at the point of partial overlapping. It preserves all the spatial information of an image by the least possible number of cuttings and the number of subparts generated by this sparse cutting mechanism are reduced significantly.

However, multiple subparts of the same symbolic object within the string might make the inference of spatial reasoning somewhat complicated. For solving the limitations of subpart management in 2D C-strings, a variation of symbolic projections called 2D B-string [6] was suggested using the interval projection method in which no cutting is required. Petraglia *et al.* [7] defined a mapping between images and indexes that satisfies the property of normalization for more freedom in the filing of images. The indexed 2D C-strings numbers the derived subparts and produces a virtual image with respect to reverse invariance. Petraglia *et al.* also introduced 2R string [7], using a polar axis system that substitutes the Cartesian one, to achieve rotation invariance. Recently, Jungert and Chang proposed the $\sigma$-tree [8], for logical description of images, which is a symbolic hierarchical representation of a 2D or 3D space such that spatial reasoning or iconic indexing can be performed.

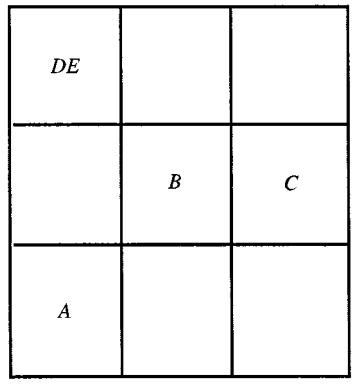
The various 2D string variants have their respective improvement or advantage in the representation or handling of spatial knowledge over 2D strings. However, solving the problem of image query or similarity retrieval of images based on 2D string subsequence matching is not suitable unless a ranking mechanism is employed. The analogous matching mechanism, as developed in 2D C-string also using a ranking scheme, can be employed for similarity retrieval for 2D string variants. The essence of the ranking mechanism is based on an implicitly implied tree structure. To solve the

problem from the root, therefore, in this paper we propose a new spatial representation called 2D C-tree for iconic images. The spatial relationships among symbolic objects are retained and the spatial knowledge is implicitly embedded in the characteristic structure of an ordered labeled tree. The spatial knowledge is derivable from a 2D C-tree simply by the common operations of tree structure. A specific tree matching algorithm can be developed to solve the problem of image retrieval and subpicture query. In the rest of the paper, the 2D C-string approach is briefly recalled in the next section. Then we present the characteristic tree structure and the construction algorithm in Section 3. The spatial reasoning of 2D C-tree is also analyzed in Section 4. In Section 5, we make a brief description and illustration of the use of 2D C-tree for image query and retrieval. Finally, conclusions are given in the last section.

## 2. Spatial indexing by 2D C-Strings

The original 2D string [1] is point-based for representing symbolic pictures. The basic idea is to project the objects of a picture along the *x*- and *y*-coordinates to form two strings representing the relative positions of objects in the *x*- and *y*-axis, respectively [4]. Three spatial relation operators '$<$', '$=$', and ':' are employed in 2D strings. The operator '$<$' denotes the 'left-right' or 'below-above' spatial relation. The operator '$=$' denotes the 'at the same spatial location as' relation. The operator ':' denotes the 'in the same set as' relation. A symbolic picture and its corresponding 2D string representation are shown in Figure 1.

However, the operators of 2D strings are not sufficient to give a complete description of spatial knowledge for images of arbitrary complexity. Chang *et al.* extended the idea of symbolic projection by introducing the cutting mechanism and proposed the generalized 2D string (2D G-string) [3] for images with overlapping objects. The cuttings are performed at all the extreme points of symbolic objects in the picture, but they are not economic for complex images in terms of storage space and navigation complexity. A more efficient cutting mechanism with a characteristic set of spatial operators as illustrated in Table 1 was introduced in the 2D C-string representation [5]. The notations $A_b$ and $A_e$ ($B_b$ and $B_e$) denote the begin- and end-bound of object $A$ ($B$), respectively.




$u : A=DE<B<C$

$v : A<B=C<DE$

Figure 1. A symbolic picture and its corresponding 2D string representation

Table 1. The definition of spatial operators

| Notation | Condition | Meaning |
|---|---|---|
| $A < B$ | $A_e < B_b$ | $A$ disjoins $B$ |
| $A \mid B$ | $A_e = B_b$ | $A$ is edge to edge with $B$ |
| $A = B$ | $A_b = B_b$ and $A_e = B_e$ | $A$ has the same projection as $B$ |
| $A [ B$ | $A_b = B_b$ and $A_e > B_e$ | $A$ contains $B$ and they have the same begin-bound |
| $A ] B$ | $A_b < B_b$ and $A_e = B_e$ | $A$ contains $B$ and they have the same end-bound |
| $A \% B$ | $A_b < B_b$ and $A_e > B_e$ | $A$ contains $B$ and they do not have the same bound |
| $A/B$ | $A_b < B_b < A_e < B_e$ | $A$ is partly overlapping with B |

Basically, the cutting of 2D C-string is performed only at the point of partial overlapping. The spatial operators are classified into two categories: global operators and local operators. The global operators '$<$' and '$|$' handle the cases of non-overlapping. The local operators '[', '%', ']', '$=$' and '$/$' handle the cases of overlapping. Since transitivity does not hold in the inference of spatial reasoning when the derivation involves the partly overlapping operator '$/$', there might incur ambiguity. The '$/$' operator can be expressed in terms of other operators and is dropped from the set of spatial operators for unique representation.

As for the case of two objects that are partly overlapping, a cutting will be performed at the end-bound point of the *preceding* object.

Definition 1. For two objects $A$ and $B$, the object $A$ is called the *preceding* object of object $B$ when $A_b < B_b$ (the begin-bound of object $A$ is smaller than that of object $B$). On the contrary, the object $B$ is called the *following* object of object $A$.

Definition 2. For the case of two objects that are partly overlapping, the *preceding* object is called the *dominating* object of the *following* object.

The cutting mechanism keeps the *preceding* object intact and partitions the *following* object. The *following* object is forced to split into two subparts. The first subpart is terminated at the end-bound of the *dominating* object, and the remaining subpart is reborn at the same location. For example, $B/C$ can be transformed to $B]C_1|C_2$, where $C_1$ denotes the first subpart of object $C$ and $C_2$ the second subpart. The cutting mechanism is also suitable for images with many objects. Consider the example image with three objects $B$, $C$, and $D$, as shown in Figure 2. Only one cutting is performed at the end-bound of the *dominating* object $B$, and then the remaining subparts of $C$ and $D$ are now completely overlapping, not partly overlapping. The example image in Figure 2 is represented as $B]C_1]D_1|D_2[C_2$. Furthermore, the cutting, which is performed at the end-bound of a *dominating* object, does not partition the *preceding* objects of this *dominating* object. A simple example is shown in Figure 3. Object $B$ is the *dominating* object of object $C$, and object $A$ is the *preceding* object of object $B$. Object $C$ is segmented by object $B$ naturally, but object $A$ is not. Thus the example image is represented as $A\%(B]C_1 \mid C_2)$.

In the example image $f$ in Figure 4, the object $A$ is partly overlapping with the objects $C$ and $D$, and the cutting is performed at the end-bound point of object $A$ along the

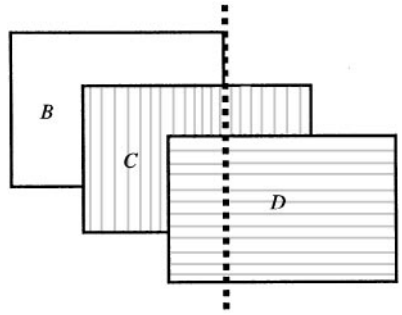



Figure 2. Object *B* is a dominating object

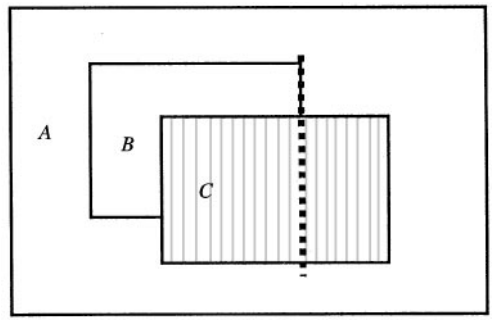



Figure 3. *B* is a dominating object of *C*, but *A* is not segmented by *B*

*x*-direction. The objects *C* and *D* are each partitioned into two subparts. The 2D C-string representation of the image *f* along the *x*-axis is

$$\text{2D C-string}(f)\text{: } A \,]\, C_1 = D_1 \,\%\, B \,|\, D_2 \,[(C_2 < E).$$

All the spatial information of an image is preserved by the least possible number of cuttings and the number of subparts generated by this sparse cutting mechanism is reduced significantly. Considering the spatial knowledge of 2D C-strings, three kinds of fundamental laws were explored from the algebraic point of view of the 2D C-string [9]. The transitive laws are capable of deriving any binary relationships among three successive objects in a 2D C-string. The distributive laws are used to handle the case of local body [5]. The manipulation laws are for the inference of the segmented objects treating their relationships integrally. It has been proved that all the binary relationships among objects in a picture can be inferred from a 2D C-string.

The problem of how to infer the spatial relations between two symbolic objects from a given 2D C-string representation in spatial reasoning is solved by using the rank mechanism [10]. The rank values of objects stand for the relative sequencing in the 2D C-string representing the relative spatial positioning of the original symbolic image. The rank plays an important role in 2D string subsequence matching [5]. The spatial knowledge is embedded in the ranks of the symbolic objects. In fact, the ranks are representative of the spatial knowledge of the symbolic objects in an image. However,

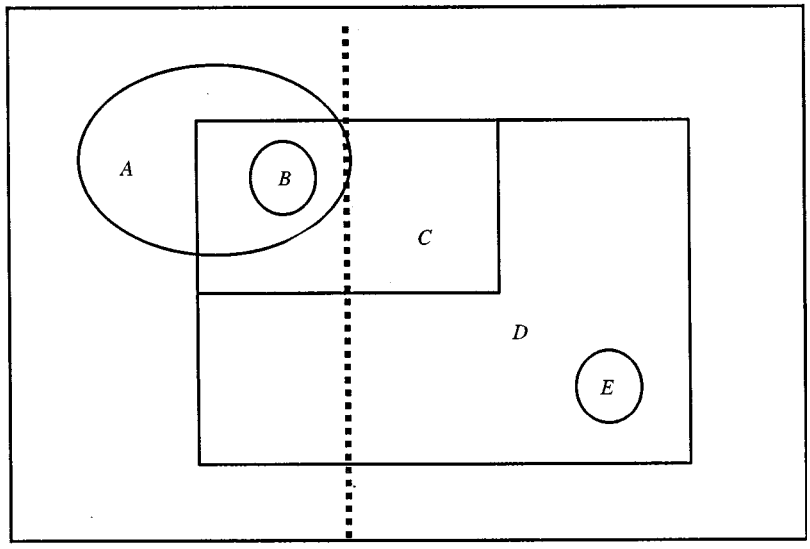



Figure 4. The image *f* with the cutting of 2D C-string representation

the computation of the rank values of objects might make the inference of spatial reasoning somewhat complicated due to the transformation of the implicitly hidden tree structure of a 2D C-string. Therefore, we directly explore a tree structure for the 2D C-string representation.

## 3. 2D C-Tree

### 3.1. 2D C-Cutting Algorithm

Suppose that the objects are recognized and enclosed by minimum bounding rectangles in a given image. Each minimum bounding rectangle has both begin-bound and end-bound along the *x*- and *y*-coordinate axis, respectively. The two orthogonal sets of cutting corresponding to the cuts along the *x*- and *y*-direction are independent of each other, and the symbolic picture is constructed into respective 2D C-tree [11] along the *x*- and *y*-axis individually. Since the spatial relationships among many objects in the image are complicated, some objects may be segmented by cutting lines to represent the spatial relationships in good order. Before detecting the cuttings, the values of all the begin-bounds and end-bounds of objects need to be sorted. Then the same-value points are grouped into the same-value sets and will be abbreviated as *sets* in the following context. The same-value *sets* are the principals of the algorithm. For the example image *f* in Figure 4, there are five objects in the image. After sorting and grouping the values of all bound points, there are nine same-value *sets* as follows.

$$\{A_b\},\ \{C_b, D_b\},\ \{B_b\},\ \{B_e\},\ \{A_e\},\ \{C_e\},\ \{E_b\},\ \{E_e\},\ \{D_e\}.$$

For each *set*, we must check whether there is any end-bound point in the *set*. The case of partly overlapping occurs only in the condition that the end-bound of a *dominating*

object exists in the *set*. From the same end-bound objects in the *set*, the object which has the smallest begin-bound is chosen to be the candidate of *dominating* object. If there is any *following* object that does not end yet, the candidate becomes a *dominating* object. The *dominating* object forces all the *following* objects either to terminate or to split into two subparts. If the end-bound of a *following* object is also in the *set* fortunately, the *following* object terminates naturally. If not, the *following* object is segmented into two subparts. The detailed cutting algorithm, called 2D C-Cutting, is described as follows.

Algorithm: 2D C-Cutting

Step 1: Sort the values of all the begin-bound and end-bound points of objects.

Step 2: For the sorted points, group the same-value points into same-value *sets*.

Step 3: Loop from step 4 to step 8 for each *set* until no more *set*, then finish.

Step 4: Check whether there is any end-bound in the current *set*. If all points in the *set* are begin-bounds, go back to step 3 for the next *set*.

Step 5: Find the *dominating* object selected from the same end-bound objects in the current *set*. The *dominating* object has the smallest begin-bound. (A cutting is performed at the location of this *set*, if the case of partly overlapping occurs.)

Step 6: If there is any *following* object whose begin-bound is in the range of the *dominating* object, cut all the *following* objects and link them. Otherwise, go to step 7.

(1) Cut the *following* objects. If the end-bound of a *following* object is also in the current *set*, the *following* object ends naturally and does not need to be cut. Otherwise, the *following* object is segmented into two subparts because the case of partly overlapping objects occurs. The front subpart is to be linked in the next substep and the back subpart will be collected into the current *set* at step 8.

(2) Link the *following* objects (or subparts). According to the begin-bound values of the *following* objects/subparts, the same begin-bound objects/subparts are linked by '=' to constitute an individual same-begin list (abbreviated as *list*). Then the *lists* are linked by ']' together from the largest to the smallest. If the smaller *list* had linked other objects (or *lists*) before, two additional actions need to be done. (a) The operator is changed to '|' when the larger *list* had been marked the 'edge' flag at step 8 previously; otherwise, the operator is changed to '<'. (b) If the operator between the smaller *list* and its first linked object/*list* is '[' or '%', it is changed to '=' or ']' correspondingly.

Step 7: If there is more than one *dominating* object with same begin- and end-bound in the current *set*, link them by '=' into a *dominating list*. If there is other same begin-bound object that does not end yet, the *dominating* object|*list* is linked to this object by '['. Otherwise, the *dominating* object*list* is linked to its *preceding* object by '%' operator. If the *preceding* object had linked other objects before, the operator is changed to '|' when the *dominating* object/*list* had been marked the 'edge' flag at step 8 previously; otherwise, the operator is changed to '<'.

Step 8: Remove the end-bound points from the current *set* and remove the begin-bound points of all the *following* objects from the previous *sets* within the range of the *dominating* object. The remaining subparts of segmented objects in step 6(1) are viewed as new objects with begin-bound being the location of the current *set* and merged into this *set*. Then mark this *set* with an 'edge' flag. If the *set* is empty, de-allocate this *set*.

Obviously, the 2D C-Cutting algorithm takes $O(N \log N)$ time for sorting and $O(N_2)$ time for parsing the objects in an image. The process of 2D C-Cutting algorithm for $f$ in Figure 4 is briefly explained and demonstrated in Figure 5. The original input data, all the begin-bound and end-bound points of objects, are shown in Figure 5(a). After sorting the values of points at step 1, the sorted points are shown in Figure 5(b) and grouped into nine same-value *sets* at step 2 as shown in Figure 5(c). At step 3, loop from step 4 to step 8 for each *set* until no more *set*.

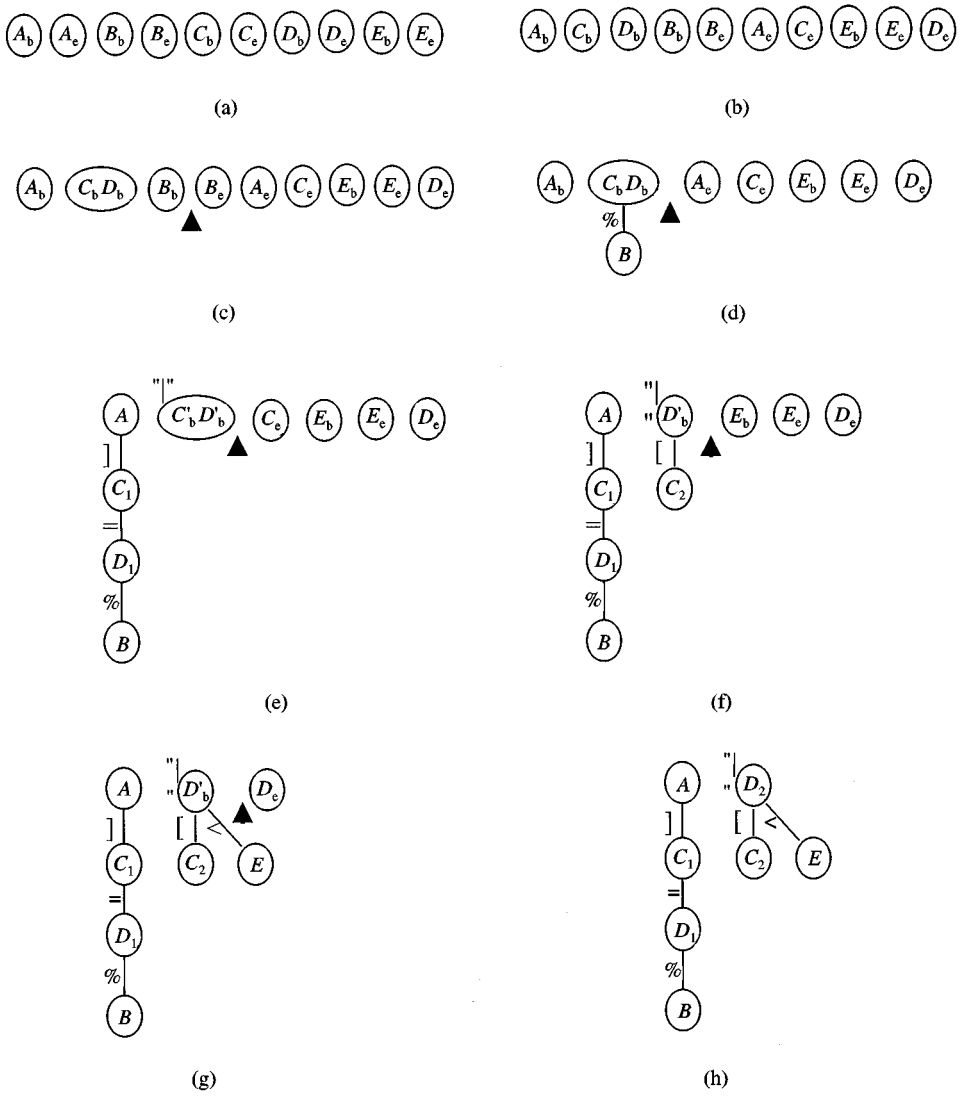



Figure 5. The construction process of 2D C-Cutting Algorithm for image *f*. (a) The original input data. (b) The sorted value points. (c) Nine same value sets. (d) Object *B* has ended. (e) Object *A* is a dominating object. (f) Object *C* has ended. (g) Object *E* keeps intact. (h) The final result

The first three *sets* are all begin-bound points, so nothing has to be done at step 4. A processing indicator currently passes over the third *set* as shown in Figure 5(c). The fourth *set* has one point $B_e$ which is an end-bound point. Since $B_e$ is the only end-bound point in the current *set*, $B$ is a *dominating* object itself (step 5). Because there is no *following* object in the range of object $B$ (step 6), $B$ has ended and is linked to its *preceding* object (the second *set*) by '%' operator (step 7). The third and fourth *sets* are de-allocated after removing $B_b$ and $B_e$ at step 8. The temporary result is shown in Figure 5(d).

The next *set* has one point $A_e$, and $A_e$ is also the only end-bound point in this *set*. Because there are two begin-bound points $C_b$ and $D_b$ in the range of object $A$, the case of partly overlapping objects occurs. Object $A$ becomes the *dominating* object of objects $C$ and $D$ (steps 5 and 6). The *following* object $C$ is segmented into two subparts $C_1$ and $C'$. Object $D$ is also segmented into two subparts $D_1$ and $D'$ (step 6(1)). The same begin-bound subparts $C_1$ and $D_1$ are linked by '=' to constitute a same-begin *list*. Then the *list* is linked to its *preceding* object (the first *set*) by ']' (step 6(2)). The *dominating* object A has ended immediately and produces the first portion of the given image because no object is preceding object $A$ (step 7). Three begin points $A_b$, $C_b$, $D_b$ and an end-bound point $A_e$ are all removed. The remaining subparts of segmented objects $C$ and $D$, i.e. $C_b'$ and $D_b'$, are collected into the current *set*, and this *set* is marked with an 'edge' flag (step 8). The temporary result is shown in Figure 5(e).

The next *set* has one point $C_e$. Since there is no *following* object in the range of the reborn object $C'$ (step 5 and step 6), the object $C'$ has ended and becomes the second subpart of $C$, $C_2$. Then $C_2$ is linked to another same begin-bound point $D_b'$ that does not end yet by '[' (step 7). Remove two points $C_b'$ and $C_e$ and de-allocate the *set* containing $C_e$ at step 8. The temporary result is shown in Figure 5(f).

The next two *sets* constitute an intact object $E$. At step 7, similar to object $B$, object $E$ has to be linked to its *preceding* object $D_b'$. Since the *preceding* object $D_b'$ has contained another object $C_2$, object $E$ is linked to $D_b'$ by '<' operator because no 'edge' flag is marked in the *set* containing $E_b$. The temporary result is shown in Figure 5(g).

Finally, the last point $D_e'$. Because there is no *following* object, the object $D'$ has ended naturally and becomes the second subpart of $D$, $D_2$. The subtree rooted at $D_2$ forms the second portion of the given image. It is noted that the 'edge' flag is marked on the object $D_2$ in the phase of Figure 5(e). The final result is shown in Figure 5(h). Now, the algorithm 2D C-Cutting is accomplished and we can construct a labeled tree, called the signed 2D C-tree.

## 3.2. Constructing a 2D C-Tree

We first initialize a root of 2D C-tree, $R$, which represents the margin or boundary of the area covered by a given image. Then all the portions produced in the process of 2D C-Cutting are sequentially linked to the root $R$. The first portion is always linked to the root by '=' operator. Other portions are linked by '|' operator if the 'edge' flag is marked on the portions; otherwise, by '<' operator. The signed 2D C-tree of $f$ is constructed as shown in Figure 6.

Each node with label represents an object or subpart in the image. The link, called the signed link, connecting two nodes is signed with the relation operator between the *preceding* objectısubpart and the following objectısubpart. For the ordered subtree rooted at node $S$ with n immediate descendants in the ordering $s_1, s_2, \ldots, s_n$, $S$ being the parent is

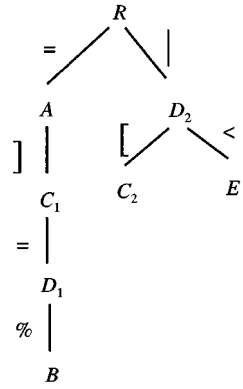



Figure 6. The signed 2D C-tree of image *f*

actually the *preceding* object of all its immediate child nodes. The relation operator between node *S* and its first child-node $s_1$ is surely a local operator that indicates the ensemble relationship between *S* and the block consisting of all its child-nodes $s_1$, $s_2, \ldots, s_n$. For example, in Figure 6, the '[' operator between $D_2$ and its first child $C_2$ indicates that $D_2$ is the same-begin-bounded overlapping with the local body consisting of $C_2$ and *E*. The relation operators between node *S* and other child-nodes $s_i$ $(2 \leq i \leq n)$ are definitely global operators that indicate the sibling relation between the child-node $s_i$ $(2 \leq i \leq n)$ and the prior child-node $s_{i-1}$ of node *S*. For example, the '$<$' operator between $D_2$ and *E*, in Figure 6, indicates the sibling relation between $C_2$ and *E*. However, a tree with signed link is somewhat unusual for general applications. And the specific tree cannot benefit from currently available tree algorithms. Consequently, a variant 2D C-tree is investigated to comply with the general usage of conventional trees.

The *empty-node* defined below is employed in order to remove the relation operator from the signed link according to the basic definition of the operators.

Definition 3. An *empty-node* is a pseudo-node which is labeled '$\varepsilon$' and can be of various size.

In the following transformation process, the example signed tree for image *f* in Figure 6 is taken for illustration and the transformed tree is depicted in Figure 7. The relation operators of the signed links can be removed by the transformation rules below.

(TR_1) The ']' operator between a node *S* and its first child-node $s_1$ can be removed by adding an *empty-node* before the first child-node $s_1$. For example, the ']' operator between *A* and $C_1$ is removed by adding $\varepsilon_1$.

(TR_2) The '%' operator between a node *S* and its first child-node $s_1$ can be removed by adding an *empty-node* before the first child-node $s_1$ and after the last child-node $s_n$, respectively. For example, the '%' operator between $D_1$ and *B* is removed by adding $\varepsilon_2$ and $\varepsilon_3$.

(TR_3) The '[' operator between a node *S* and its first child-node $s_1$ can be removed by adding an *empty-node* after the last child-node $s_n$. For example, the '[' operator between $D_2$ and $C_2$ is removed by adding $\varepsilon_5$.

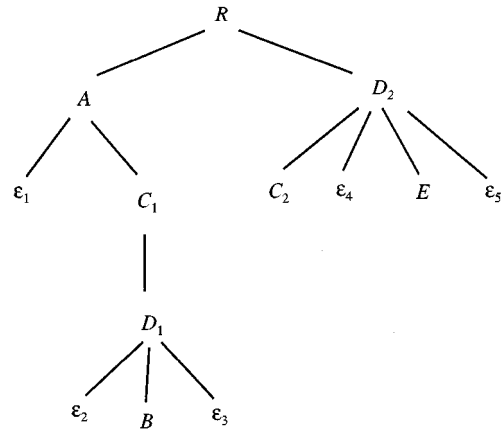



Figure 7. The transformed 2D C-tree with operators removed

(TR_4) The '=' operator between a node $S$ and its first child-node $s_1$ can be removed unconditionally. For example, the '=' operator between the root $R$ and the object A can be removed. Also, the '=' operator between $C_1$ and $D_1$ is removed.

(TR_5) The '|' operator between a node $S$ and its child-node $s_i$ $(2 \leq i \leq n)$ can be removed unconditionally. For example, the '|' operator between $R$ and $D_2$ can be removed.

(TR_6) The '<' operator between a node $S$ and its child-node $s_i$ $(2 \leq i \leq n)$ can be removed by inserting an *empty-node* before this child-node $s_i$. For example, the '<' operator between $D_2$ and $E$ is removed by adding $\varepsilon_4$.

When the relation operators are stripped off from a signed 2D C-tree after employing the above transformation rules, each node of the transformed tree has at least two child-nodes except the node that originally connects to its single child-node by removing the '=' operator. The '=' operator possesses the commutation law, which is different from other relation operators of 2D C-tree. The objects which are connected with the '=' operator have the same begin-bound and end-bound. For example, the '=' operator between $C_1$ and $D_1$ indicates that these two objects/subparts have the same begin-bound and end-bound. For the reasoning of relationship among the nodes of 2D C-tree, a special *set-node* is introduced for treating a set of lineage that each node has single child-node.

Definition 4. A *set-node* is a multi-label node consisting of objects that have the same begin-bound and end-bound.

Thus, an additional transformation rule is proposed.

(TR_7) A node $S$ that has single child-node can be merged with its single child-node to form a *set-node* and the descendants of the child-node become the

descendants of the *set-node*. For the example in Figure 7, $C_1$ is merged with its single child-node $D_1$ to form a *set-node* $\{C_1, D_1\}$ and the descendants of $D_1$, $\varepsilon_2$, B, and $\varepsilon_3$ become the descendants of the *set-node* $\{C_1, D_1\}$ as shown in Figure 8.

Additionally, many properties of 2D C-trees are pointed out.

Lemma 3.1. *Any two nodes of a 2D C-tree are either completely overlapping or non-overlapping.*

Proof. Since the cuts are performed at the cases of partly overlapping objects, any object or segmented subpart is exempt from partly overlapping with the others. □

Lemma 3.2. *An* empty-node *of a 2D C-tree is a leaf node.*

Proof. The *empty-node* is a pseudo node for removing the relation operator between nodes. Clearly, it is impossible that any substantial node is a child-node of an *empty-node*. □

Lemma 3.3. *Each internal node of a 2D C-tree has at least two child-nodes.*

Proof. Considering the case that the internal node has only one child-node in a signed 2D C-tree, the internal node is definitely completely overlapping its child-node. Either these two nodes are merged to a *set-node* by using (TR_4) and (TR_7) for the case that the internal node is originally connected to its child-node with '=' operator, or some *empty-nodes* are added by using (TR_1)–(TR_3) for the cases that the internal node is originally connected to its child-node with ']', '%', or '[' operator. The internal node becomes a *set-node* in the former case. And the internal node has at least two child-nodes in the latter case. □

By employing the transformation rules from (TR_1) to (TR_7), a general 2D C-tree is obtained and is in conformity with the above properties. The sample symbolic image *f* in Figure 4 is represented in a general 2D C-tree as shown in Figure 8 with subscript of *empty-nodes* being omitted.

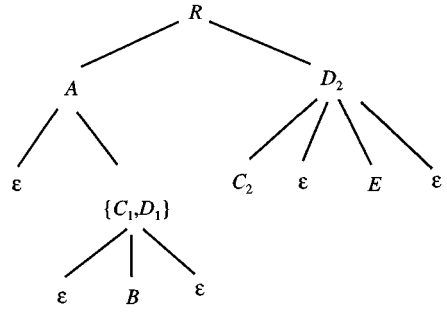



Figure 8. The general 2D C-tree of image *f*

## 4. Spatial Reasoning Using 2D C-Tree

The 2D C-tree is an ordered labeled tree in which each node has a label and the left-to-right order of its children (if it has any) is fixed. Before the discussion of spatial reasoning on 2D C-tree, the notations on trees used [12] are briefly described and are listed in the Notation section.

(1) *Dewey decimal notation:* Let $\mathfrak{I}$ be a rooted tree. The Dewey decimal notation (abbreviated D-notation) of the root of the tree is '0'. Assume that '$\alpha$' is a D-notation of a node in $\mathfrak{I}$, and the node has $n$ immediate descendants. Then $\alpha.1, \alpha.2, \ldots, \alpha.n$ are the D-notations of the $n$ immediate descendants from left to right, respectively. The D-notation of the 2D C-tree for the image $f$ is shown in Figure 9.
(2) *Subtree:* A subtree that is rooted at $\alpha$ in $\mathfrak{I}$ is denoted by $\alpha/\mathfrak{I}$.
(3) *Size:* The size of $\mathfrak{I}$ is the number of nodes in $\mathfrak{I}$, denoted by $|\mathfrak{I}|$.
(4) *Label:* For $\alpha \in \mathfrak{I}$, $S_{\mathfrak{I}}(\alpha)$ denotes the label of $\alpha$.
(5) *Depth:* For $\alpha \in \mathfrak{I}$, the depth of $\alpha$, denoted by $D_{\mathfrak{I}}(\alpha)$, is the number of nodes on the path from the root of $\mathfrak{I}$ to $\alpha$, excluding $\alpha$.
(6) *Level:* For a subtree rooted at $\alpha \in \mathfrak{I}$, $L_{\mathfrak{I}}(\alpha)$ denotes the largest depth among all its descendants.
(7) *Predecessor:* For $\alpha \in \mathfrak{I}$, $P_{\mathfrak{I}}(\alpha)$ denotes the set of predecessors of $\alpha$. Also, $P^i_{\mathfrak{I}}(\alpha)$ denotes the $i$th predecessor of $\alpha$, where $0 < i \leq D_{\mathfrak{I}}(\alpha)$. $\alpha$ is the root of $\mathfrak{I}$ if $p_{\mathfrak{I}}(\alpha) = \emptyset$.
(8) *Cardinal number:* For $\alpha \in \mathfrak{I}$, the cardinal number of $\alpha$ is the number of immediate descendants of $\alpha$, denoted by $C_{\mathfrak{I}}(\alpha)$. $\alpha$ is a leaf node if $C_{\mathfrak{I}}(\alpha) = 0$.
(9) *Legitimate descendants:* For $\alpha \in \mathfrak{I}$, $p_{\mathfrak{I}}(\alpha)$ denotes the set of legitimate descendants of $\alpha$. Also, $p^i_{\mathfrak{I}}(\alpha)$ denotes the $i$th legitimate descendant of $\alpha$, that is, $p^1_{\mathfrak{I}}(\alpha) = \alpha.1$. $p^i_{\mathfrak{I}}(\alpha) = p^{i-1}_{\mathfrak{I}}(p^1_{\mathfrak{I}}(\alpha))$, where $1 \leq i \leq L_{\mathfrak{I}}(\alpha)$. $p_{\mathfrak{I}}(\alpha) = \emptyset$ if $\alpha$ is a leaf node.
(10) *Lateral descendants:* For $\alpha \in \mathfrak{I}$, $q_{\mathfrak{I}}(\alpha)$ denotes the set of lateral descendants of $\alpha$. Also, $q^i_{\mathfrak{I}}(\alpha)$ denotes the $i$th lateral descendant of $\alpha$, that is, $q^1_{\mathfrak{I}}(\alpha) = \alpha.n$, where $n = C_{\mathfrak{I}}(\alpha)$. $q^i_{\mathfrak{I}}(\alpha) = q^{i-1}_{\mathfrak{I}}(q^1_{\mathfrak{I}}(\alpha))$, where $1 \leq i \leq L_{\mathfrak{I}}(\alpha)$. $q_{\mathfrak{I}}(\alpha) = \emptyset$ if $\alpha$ is a leaf node.

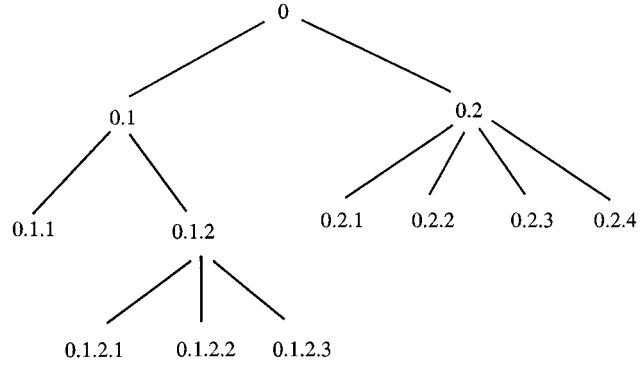



Figure 9. The D-notation of the 2D C-tree for image $f$

(11) *Postfix ordering:* The postfix ordering of nodes in $\mathfrak{I}$ is a one-to-one mapping function $f_{\mathfrak{I}}$: $\alpha \rightarrow \varpi$, where $\alpha$ is a D-notation of a node in $\mathfrak{I}$ and $\varpi \in \{1, 2, \ldots, |\mathfrak{I}|\}$, that satisfies the following conditions:

(i) $f_{\mathfrak{I}}(\alpha.i) < f_{\mathfrak{I}}(\alpha)$, where $1 \leq i \leq C_{\mathfrak{I}}(\alpha)$.
(ii) $f_{\mathfrak{I}}(\alpha.i) < f_{\mathfrak{I}}(\alpha.j)$, where $1 \leq i < j \leq C_{\mathfrak{I}}(\alpha)$.
(iii) $f_{\mathfrak{I}}(\alpha.i) < f_{\mathfrak{I}}(\alpha.j.k)$, where $1 \leq i < j \leq C_{\mathfrak{I}}(\alpha)$ for all D-notation $k$ in subtree $\alpha.j/\mathfrak{I}$.

The postfix ordering of the 2D C-tree for the image $f$ is shown in Figure 10.

(12) *Postfix representation:* Let $\alpha$ be a node in tree $\mathfrak{I}$ for which $C_{\mathfrak{I}}(\alpha) = n$. The subtree $\alpha/\mathfrak{I}$ can be represented recursively by $\alpha/\mathfrak{I} = \alpha\,(\alpha.1/\mathfrak{I}, \alpha.2/\mathfrak{I}, \ldots, \alpha.n/\mathfrak{I})$, when $n > 0$, and $\alpha/\mathfrak{I} = \alpha$ when $n = 0$.

(13) *Leading descendant:* For $\alpha \in \mathfrak{I}$, $\theta_{\mathfrak{I}}(\alpha)$ denotes the node with smallest postfix ordering of subtree $\alpha/\mathfrak{I}$.

Suppose that $X$ and $Y$ are two objects in a 2D C-tree $\mathfrak{I}$. Their nodes in D-notation are $\alpha$ and $\beta$, and the labels of these nodes are $S_{\mathfrak{I}}(\alpha)$ and $S_{\mathfrak{I}}(\beta)$, respectively. Assume that the postfix ordering of $\alpha$ is after that of $\beta$, that is, $f_{\mathfrak{I}}(\beta) \leq f_{\mathfrak{I}}(\alpha)$. Many inference rules are obtained according to the definition of relation operators.

(IR_1) $X$ and $Y$ are completely overlapping if and only if $\beta$ is in the subtree rooted at $\alpha$, i.e. $\beta \in \alpha/\mathfrak{I}$. Besides, $\beta \in \alpha/\mathfrak{I}$ if and only if $f_{\mathfrak{I}}(\theta_{\mathfrak{I}}(\alpha)) \leq f_{\mathfrak{I}}(\beta) \leq f_{\mathfrak{I}}(\alpha)$. The four succinct rules below are for the four cases of completely overlapping.

(IR_2) $X = Y$ if and only if $\alpha = \beta$. For example, $C_1$ and $D_1$ are contained in the multi-label *set-node* $\{C_1, D_1\}$ in Figure 8. $C_1 = D_1$ since they have same D-notations.

(IR_3) $X\,[\,Y$ if and only if $\beta \in p_{\mathfrak{I}}(\alpha)$. For example, $D_2\,[\,C_2$ in Figure 8.

(IR_4) $X\,]\,Y$ if and only if $\beta \in q_{\mathfrak{I}}(\alpha)$. For example, $A\,]\,C_1$ in Figure 8.

(IR_5) $X\,\%\,Y$ if and only if $\beta \in \alpha/\mathfrak{I}$, but not in the above three cases. For example, $A\,\%\,B$ in Figure 8.

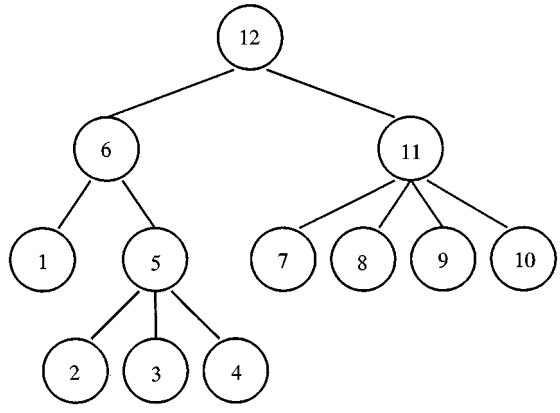



Figure 10. The postfix ordering of the 2D C-tree for image $f$

If $\beta$ is not in the subtree rooted at $\alpha$, i.e. $\beta \notin \alpha/\Im$, $Y$ is surely non-overlapping with $X$. There are two cases of relation between objects $Y$ and $X$. One is that $Y$ is edge to edge with $X$, and the other is that $Y$ disjoins $X$. The first case occurs when $\beta$ and $\alpha$ are individually contained in two *'adjacent'* subtrees.

Definition 5. Two subtrees are *adjacent* if the root of one subtree is immediately followed by the leading descendant of the other subtree according to the postfix ordering. These two subtrees are called *adjacent* subtrees.

Moreover, the case that $\beta$ is a lateral descendant of the *preceding adjacent* subtree and $\alpha$ is a legitimate descendant of the *following adjacent* subtree, implies that $Y$ is edge to edge with $X$. Otherwise, $Y$ disjoins $X$.

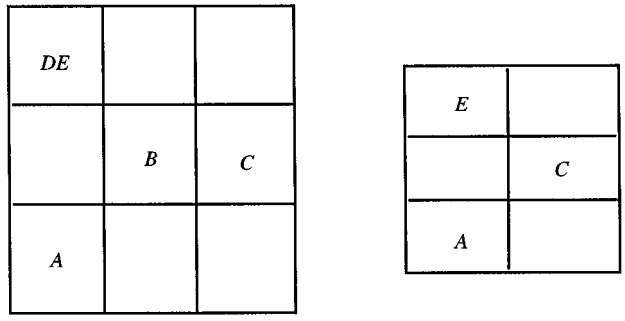
(IR_6) $Y \mid X$ if and only if $\beta \notin \alpha/\Im$ and $\beta \in q_\Im(f_\Im^{-1}(f_\Im(\theta_\Im(\alpha)) - 1))$, where $f^{-1}$ is the reverse function of $f$. For example, $A \mid D_2$ in Figure 8.

(IR_7) $Y < X$ if and only if $\beta \notin \alpha/\Im$ and not the case of $Y \mid X$. For example, $A < E$ in Figure 8.

The above rules can be examined by the tree traversal algorithm taking $O(N)$ time complexity for $N$ nodes and they are useful for inferring the spatial relationships among objects in a 2D C-tree.

## 5. Image Retrieval by 2D C-trees Matching

For many applications, we often deal with the problem of 'finding images with similar spatial relations as a query example' in image information systems. The target of image retrieval is to retrieve the images that are most similar to the query image. The similarity based upon the minimum-distance criterion has been proposed using the techniques of 2D string matching defined in terms of the longest common subsequence [1]. For example, the symbolic picture $f_1$ in Figure 11(a) may be represented as the 2D string




(a) $f_1$ (b) $q_1$

Figure 11. A symbolic image $f_1$ and a query $q_1$

($A = DE < C,\ A < B = C < DE$). We may want to retrieve images satisfying a certain query $q_1$ as in Figure 11(b). Also, $q_1$ can be translated into the 2D string ($A = E < C, A < E$). This query string is the sub-string of the 2D string representation of the image $f_1$. The problem of image retrieval then becomes the problem of 2D string subsequence matching.

Although the 2D C-string is more efficient in representation and manipulation of images, it is not suitable for solving the image query based on 2D string subsequence matching. For example, we use a query $q_2$ in order to match the picture $f_2$ as in Figure 12. The 2D C-string representation of the picture $f_2$ and $q_2$ are $(A]D[E\,|\,B\,|\,C,\ A\,|\,B\%C\,|\,D\%E)$ and ($A \,\%\, E < C, A < C < E$), respectively. The query $q_2$ is also the sub-picture of the picture $f_2$, but the query string of $q_2$ is quite different in the format from the example string of $f_2$ due to the spatial operators. The query string is not the sub-string of the example string any longer. Unfortunately these operators are needed to handle the global and local relations among symbolic objects in a 2D C-string and cannot be omitted.

A specific tree matching algorithm is required to solve the problem of image retrieval and subpicture query [13]. We use the example picture $f_2$ in Figure 12(a) and query $q_2$ in Figure 12(b) to demonstrate the subpicture query. The 2D C-trees of $f_2$ and $q_2$ along the *x*- and *y*-coordinate axis are in Figures 13 and 14 respectively. Obviously, the 2D C-trees of $q_2$ are the subtrees of the 2D C-trees of $f_2$. That is, $q_2$ is a subpicture of $f_2$.

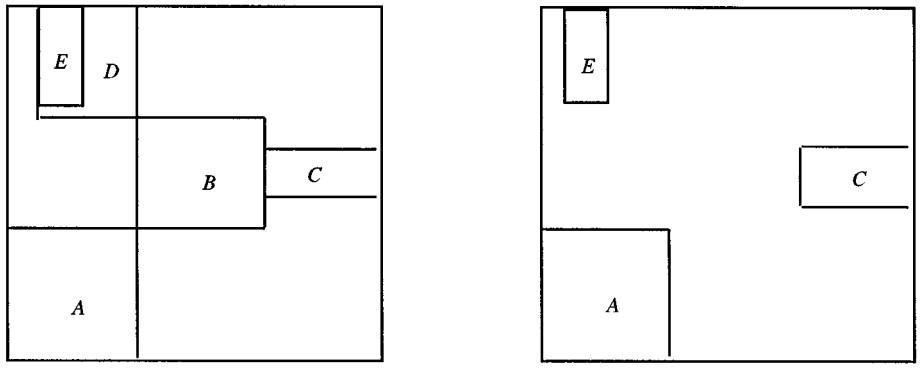



Figure 12. A symbolic image $f_2$ and a query $q_2$

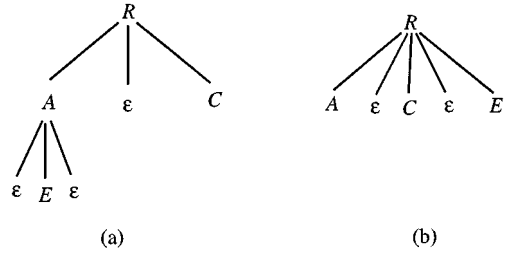



Figure 13. The 2D C-trees of query $q_2$. (a) $q_{2X}$ on *x*-coordinate axis. (b) $q_{2Y}$ on *y*-coordinate axis

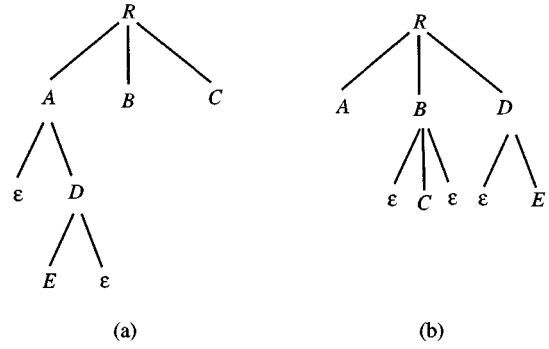



Figure 14. The 2D C-trees of image $f_2$. (a) $f_{2X}$ on $x$-coordinate axis. (b) $f_{2Y}$ on $y$-coordinate axis

We apply this new spatial representation and implement an interactive video information system [14]. The system allows users to draw a sequence of query frames by assembling object icons and computes the tree distances between the query frames and sample image frames. While all the sequences of the database are compared with the query sequence, the sequence with the minimum distance is the most similar sequence for the query.

## 6. Conclusions

The approach of 2D strings opens a novel area for iconic image indexing and retrieval. Although the 2D C-string is more efficient in representation and manipulation of images, the computation of object ranks in a 2D C-string might make the inference of spatial reasoning somewhat complicated. Moreover, the 2D C-string representation is not suitable for image query based on 2D string subsequence matching. In this paper, we propose a characteristic 2D C-tree representation with the sparse cutting mechanism, 2D C-Cutting. The new representation not only keeps the whole spatial relationships among objects in an original image, but also the spatial knowledge embedded in the characteristic structure of the 2D C-tree can be derived using the inference rules. Owing to the embedded knowledge in the tree structure, the tree matching algorithm for 2D C-tree representation is worthy of exploring to solve the problem of image retrieval in image database systems. We also apply this new spatial representation and implement an interactive video information system for similarity retrieval of video sequence in our further research.

## References

1. S. K. Chang, Q. Y. Shi & C. W. Yan (1987) Iconic indexing by 2-D Strings. *IEEE Transaction on Pattern Analysis and Machine Intelligent*, PAMI-9, 413–428.
2. E. Jungert (1988) Extended Symbolic Projection used in a knowledge structure for spatial reasoning. In: *The 4th BPRA Conference on Pattern Recognition*. Springer Verlag, Cambridge.

3. S. K. Chang, E. Jungert & Y. Li (1989) Representation and retrieval of symbolic pictures using generalized 2D strings. In: *SPIE Prococeedings on Visual Communications and Image Processing*, Philadelphia, pp. 1360–1372.
4. S. K. Chang & E. Jungert (1996) *Symbolic Projection for Image Information Retrieval and Spatial Reasoning*. Academic Press, London, U.K.
5. S. Y. Lee & F. J. Hsu (1990) 2D C-String: a new spatial knowledge representation for image database systems. *Pattern Recognition*, 23, 1077–1087.
6. S. Y. Lee, M. C. Yang & J. W. Chen (1992) Signature File as Spatial filter for iconic image database. *Journal of Visual Language and Computing* 3, 373–397.
7. G. Petraglia, M. Sebillo, M. Tucci & G. Tortora (1993) Towards normalized iconic indexing. In: *Proceedings of IEEE Sympasium on Visual Languages*, Bergen, Norway, pp. 392–394.
8. E. Jungert & S. K. Chang (1992) The $\sigma$-tree — a symbolic spatial data model. In: *Proceedings of the 11th International Conference on Pattern Recognition*. The Hague, The Netherlands, pp. 461–465.
9. S. Y. Lee & F. J. Hsu (1991) Picture algebra for spatial reasoning of iconic images represented in 2D C-String. *Pattern Recognition Letters*, 12, 425–435.
10. S. Y. Lee & F. J. Hsu (1992) Spatial reasoning and similarity retrieval of images using 2D C-String knowledge representation. *Pattern Recognition*, 25, 305–318.
11. F. J. Hsu, S. Y. Lee & P. S. Lin (1997) 2D C-tree spatial representation for iconic image. In: *Proceedings of the 2nd International Conference on Visual Information Systems*, San Diego, CA, pp. 287–294.
12. S. Y. Lu (1984) A tree-matching algorithm based on node splitting and merging. *IEEE Transaction on Pattern Analysis and Machine Intelligent*, PAMI-6, 249–256.
13. F. J. Hsu, S. Y. Lee & P. S. Lin (1998) Similarity retrieval by 2D C-Trees matching in image databases. *Journal of Visual Communication and Image Representation*, 9, 87–100.
14. F. J. Hsu, S. Y. Lee & P. S. Lin (1998) Video data indexing by 2D C-Trees. *Journal of Visual Language and Computing*, to appear.
15. S. K. Chang, E. Jungert & G. Tortora (1996) Spatial reasoning, image indexing and retrieval using symbolic projections. In: *Intelligent Image Database Systems* (S. K. Chang, E. Jungert & G. Tortora, (eds.) World Scientific, Singapore, pp. 1–8.
16. M. Flickner, H. Sawhney, W. Niblack J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele & P. Yanker (1995) Query by image and video content: the QBIC system. *IEEE Computer* 44, 23–32.
17. A. Guttman (1984) R-trees: a dynamic index structure for spatial searching. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, Boston, MA, pp. 47–57.
18. S. Y. Lee, M. K. Shan & W. P. Yang (1989) Similarity retrieval of iconic image database. *Pattern Recognition*, 22, 675–682.
19. H. Samet (1990) *The Design and Analysis of Spatial Data Structure* Addison-Wesley, Reading, MA.