Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Pattern Recognition Letters 24 (2003) 3015–3026
Pattern Recognition Letters
www.elsevier.com/locate/patrec

2D Z-string: A new spatial knowledge representation for image databases

Anthony J.T. Lee *, Han-Pang Chiu

Department of Information Management, National Taiwan University, No. 1, Section 4, Roosevelt Road, Taipei 106, Taiwan, ROC

Received 18 February 2003; received in revised form 19 June 2003

Abstract

The knowledge structure called the 2D C^+-string, proposed by Huang et al., to represent symbolic pictures allows a natural way to construct iconic indexes for images. According to the cutting mechanism of the 2D C^+-string, an object may be partitioned into several subparts. The number of partitioned subparts is bounded to $O(n^2)$, where n is the number of objects in the image. Hence, the string length is also bounded to $O(n^2)$. In this paper, we propose a new spatial knowledge representation called the 2D Z-string. Since there are no cuttings between objects in the 2D Z-string, the integrity of objects is preserved and the string length is bounded to $O(n)$. Finally, some experiments are conducted to compare the performance of both approaches.

Keywords: Iconic indexing; Image database; 2D Z-string; 2D C-string; 2D C^+-string

1. Introduction

In image database management systems, one of the most important methods for discriminating the images is using the objects and the spatial relations that exist between the objects. Hence, how images are stored in a database becomes an important design issue of image database management systems.

Over the last decade, many iconic indexing approaches to represent symbolic images have been proposed, for example, 2D string (Chang and Jungert, 1986, 1987), 2D G-string (Jungert, 1988; Jungert and Chang, 1989), 2D C-string (Lee and Hsu, 1990, 1991, 1992), 2D C^+-string (Huang and Jean, 1994), unique-ID-based matrix (Chang et al., 2000), GPN matrix (Chang et al., 2001), virtual image (Petraglia et al., 2001) and BP matrix (Chang et al., 2003). Based on those representations, several algorithms in similarity retrieval and spatial reasoning have been proposed. Those proposed algorithms can allow a user to retrieve the images similar to a query image with a specified spatial relationship.

Chang and Jungert (1986, 1987) proposed the concept of the 2D string to represent the spatial relations between the objects in an image. However, this representation is under challenge in solving the problems of spatial reasoning and planning in many applications. The main reason is

* Corresponding author. Tel.: +886-2-23630231; fax: +886-2-23621327.
E-mail address: jtlee@ccms.ntu.edu.tw (A.J.T. Lee).

0167-8655/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0167-8655(03)00162-4

that the spatial operators of 2D strings are not sufficient enough to give a complete description for a complex picture. Jungert (1988) and Jungert and Chang (1989) introduced some local operators as compensation for handling more types of relations between pictorial objects in query reasoning. Chang et al. (1988) introduced the generalized 2D string (2D G-string) with the cutting mechanism. In the 2D G-string knowledge representation, the cuttings are performed at all the end points (including the begin-bound and end-bound points) of all the objects in an image. Obviously, the cutting mechanism may result in a large number of subparts.

To reduce the number of cuttings and generated subparts, Lee and Hsu (1990, 1991, 1992) proposed the 2D C-string representation based on a special cutting mechanism. The cutting mechanism of the 2D C-string is better than that of the 2D G-string. In the 2D C-string, they introduced seven spatial operators to describe 13 possible relations between two objects as shown in Table 1, where Begin(A) and End(A) are the begin-bound and end-bound of object A.

Although the cutting mechanism of the 2D C-string is better, it still generates many subparts of objects in the case of the complex images. Suppose that there are n objects in an image. According to the analysis in (Lee and Hsu, 1990), in the worst case of the 2D G-string, each object may be partitioned at the begin-bound points and/or end-bound points of the other objects. Since the total number of cutting lines is at most $2n$, the total number of segmented subparts of partitioned objects is bounded to $O(n^2)$. As for the 2D C-string, the cuttings are performed at the end-bound points of overlapping objects. The number of cuttings in the 2D C-string is equal to the number of overlapping objects. The number of overlapping objects is bounded to n. So the order of the total number of partitioned subparts is still bounded to $O(n^2)$.

Table 1
The definitions of spatial operators in 2D C-string

Notations	Conditions
$A < B$	$\text{End}(A) < \text{Begin}(B)$
$A = B$	$\text{Begin}(A) = \text{Begin}(B)$, $\text{End}(A) = \text{End}(B)$
$A \mid B$	$\text{End}(A) = \text{Begin}(B)$
$A \% B$	$\text{Begin}(A) < \text{Begin}(B)$, $\text{End}(A) > \text{End}(B)$
$A [B$	$\text{Begin}(A) = \text{Begin}(B)$, $\text{End}(A) > \text{End}(B)$
$A] B$	$\text{Begin}(A) < \text{Begin}(B)$, $\text{End}(A) = \text{End}(B)$
A / B	$\text{Begin}(A) < \text{Begin}(B) < \text{End}(A) < \text{End}(B)$

Since the 2D C-string ignores the information about the locations, sizes of objects, and distances between objects, the 2D C-string representation may result in ambiguity. To resolve the ambiguity problem, Huang and Jean (1994) proposed the 2D C$^+$-string. In the 2D C$^+$-string representation, each object in a given image has two pairs of begin-bounds and end-bounds. One is for the projection onto the x-axis and the other is for the projection onto the y-axis. From the begin-bounds and end-bounds of the projections, the sizes of objects and the distances between objects can be calculated. It only has to calculate three kinds of metric information: the size of object $A = \text{End}(A) - \text{Begin}(A)$, the distance associated with operator "<" in "$A < B$" $= \text{Begin}(B) - \text{End}(A)$, the distance associated with operator "%" in "$A \% B$" $= \text{Begin}(B) - \text{Begin}(A)$. By using the 2D C$^+$-string representation, the ambiguity problem in the 2D C-string can be resolved.

Although the 2D C$^+$-string resolves the ambiguity problem, it uses the same cutting mechanism as the 2D C-string. So the number of partitioned subparts and the length of the string are still bounded to $O(n^2)$.

Therefore, in this paper we propose a new spatial knowledge representation for an image with a zero-cutting mechanism, called the 2D Z-string. Since there are no cuttings between objects in the 2D Z-string, the integrity of objects is preserved and the string length is bounded to $O(n)$. Therefore, the 2D Z-string is more compact and efficient than previous approaches in terms of storage space and execution time.

The rest of this paper is organized as follows. In Section 2, we present the new spatial knowledge representation of the 2D Z-string. The string generation algorithm of the 2D Z-string is described in Section 3. In Section 4, we propose the image reconstruction algorithm based on the 2D Z-string representation. In Section 5, some experiments are conducted to compare our approach with the 2D C$^+$-string approach. Finally, concluding remarks are made in Section 6.

2. 2D Z-string

In the 2D Z-string, the objects in an image are projected onto the x- (or y-)axis to form a string to represent the spatial relations between the projections. The projections onto the x-axis are called x-projections. The projections onto the y-axis are called y-projections. In the 2D Z-string representation, we record the information about the locations and sizes of objects, and the distances between objects.

There are 13 possible relations between two x- (or y-)projections in the 2D Z-string. The corresponding spatial operators of those relations have been listed in the 2D C-string as shown in Table 1. To avoid the ambiguity, operator "/" is not used in the 2D C-string. So, A/B is replaced with $A]B|B$. That is, it is required to perform a cutting for the partly overlapping objects in the 2D C-string. However, we do use operator "/" in the 2D Z-string. So no cuttings are required.

Now, we propose the knowledge structure of 2D Z-string to represent spatial relations between the objects of interest.

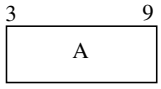
Definition 1. The knowledge structure of 2D Z-string is a 5-tuple (O, A, R_g, R_l, "( )") where

(1) O is the set of objects of interest;
(2) A is the set of attributes to describe the objects in O;
(3) $R_g = \{$"<", "|"$\}$ is the set of global relation operators;
(4) $R_l = \{$"=", "[", "]", "%", "/"$\}$ is the set of local relation operators;
(5) "( )" is a pair of separators which is used to describe a set of objects as a template object.

Then we add some metric measurements to the knowledge structure of 2D Z-string.

(1) The size of an object: A_s denotes that the size (length) of the x- (or y-)projection of object A is equal to s, where $s = \mathrm{End}_x(A) - \mathrm{Begin}_x(A)$ (or $s = \mathrm{End}_y(A) - \mathrm{Begin}_y(A)$), $\mathrm{Begin}_x(A)$ and $\mathrm{End}_x(A)$ are the x coordinates of the begin-bound and end-bound of the x-projection of object A, respectively.

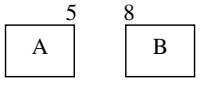
For example,



is represented by A_6.

(2) The distance associated with operator "<": $A <_d B$ denotes that the distance between the x- (or y-)projection of object A and that of object B is equal to d, where $d = \mathrm{Begin}_x(B) - \mathrm{End}_x(A)$ (or $d = \mathrm{Begin}_y(B) - \mathrm{End}_y(A)$).

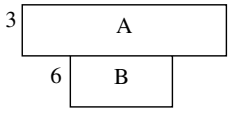
For example,



is represented by $A <_3 B$.

(3) The distance associated with operator "%": $A\%_d B$ denotes that the distance between the x- (or y-)projection of object A and that of object B is equal to d, where $d = \mathrm{Begin}_x(B) - \mathrm{Begin}_x(A)$ (or $d = \mathrm{Begin}_y(B) - \mathrm{Begin}_y(A)$).

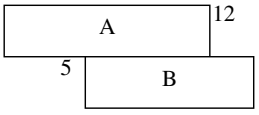
For example



is represented by $A\%_3 B$.

(4) The distance associated with operator "/": $A/_d B$ denotes that the distance between the x- (or y-)projection of object A and that of object B is equal to d, where $d = \mathrm{End}_x(A) - \mathrm{Begin}_x(B)$ (or $d = \mathrm{End}_y(A) - \mathrm{Begin}_y(B)$).

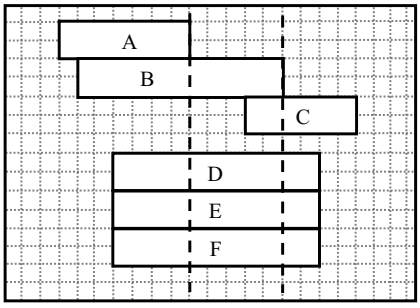
For example



is represented by $A/_7 B$.

(5) The other operators: no metric measurements.

Now, let us consider the example as shown in Fig. 1. The image shown in Fig. 1(a) contains six objects: A, B, C, D, E, and F. To generate the 2D



(a)

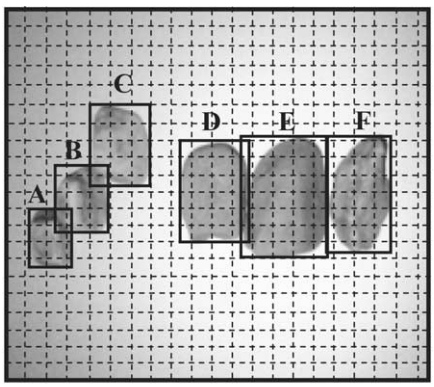
2D C^+ u-string: $A_7]B_6]D_4=E_4=F_4|B_5=D_5=E_5=F_5]C_2|C_4[D_2=E_2=F_2$

2D Z u-string: $(A_7/_6(B_{11}/_9((D_{11}=E_{11}=F_{11})/_4C_6)))$

(b)

Fig. 1. 2D C^+-string and 2D Z-string: (a) the cuttings performed along the x-axis direction in the 2D C^+-string and (b) the corresponding 2D C^+ and 2D Z u-strings.

C^+ u-string, two cuttings are performed. One is at the end-bound of object A. The other is at the end-bound of object B. Hence, objects B and C are partitioned into two subparts. Objects D, E, and F are partitioned into three subparts. The corresponding 2D C^+ and 2D Z u-strings are shown in Fig. 1(b). The 2D C^+ u-string is obviously longer than the 2D Z u-string.



(a)

u-string: $(((A_2/_{0.9}B_{2.7})/_{0.9}C_3)<_{1.4}((D_{3.4}/_{0.3}E_{4.2})|F_3))$

v-string: $((((A_{3.5}/_{1.5}B_4)/_{4.5}D_6)/_7(E_{7.2}]F_7))/_{2.8}C_{4.6})$

(b)

Fig. 2. The 2D Z-string representation for the partly overlapping objects: (a) the objects are approximated by the MBRs and (b) the corresponding 2D Z-string.

Next, let us consider another example as shown in Fig. 2. In this example, the image contains six partly overlapping objects. All the objects are approximated by the MBRs as shown in Fig. 2(a). The corresponding 2D Z-string of the image is shown in Fig. 2(b).

Therefore, we have shown that the knowledge structure of 2D Z-string can easily represent the spatial relations between objects. Since there are no cuttings between the objects in an image, the 2D Z-strings are much shorter while still preserving the spatial relations between the objects. The knowledge structure of 2D Z-string provides us an easy way to represent the spatial relations between the objects in image database management systems.

3. String generation algorithm

This section describes the *string generation algorithm* based on the zero-cutting mechanism for the 2D Z-string. The *string generation algorithm* is extended from the concept of the *string generation algorithm* proposed by Huang and Jean (1994). The major differences include: (1) in the 2D Z-string, there are no cuttings between objects; (2) the length of the 2D Z-string generated by the *string generation algorithm* is bounded to $O(n)$, where n is the number of objects in the image.

First of all, we introduce a new type of objects: template object. A template object covers the objects enclosed between separators "(" and ")" and is viewed as a new object. The begin-bound of the template object is the smallest begin-bound in all the covered objects. Similarly, the end-bound of the template object is the largest end-bound in all the covered objects.

The input to the *string generation algorithm* is a list of objects, where each object is expressed by its x- (or y-)projection $O_i(B_i, E_i)$, B_i is the begin-bound and E_i is the end-bound of the x- (or y-)projection of object O_i. To generate the u- (or v-)string, we first find all the dominating objects, which are defined by Lee and Hsu (1991). The projections of the objects with the same end-bound are grouped into a list. In the list, the object with the smallest begin-bound is called the dominating object.

By scanning from left to right along the x- (or y-)axis, we shall find all the dominating objects. For each dominating object, if there exist objects that partly overlap with the dominating object, choose among them the object with the smallest end-bound. Let the chosen object be P. If there exist no objects partly overlapping with the dominating object, find the objects covered by the dominating object (including the dominating object itself). Otherwise, find the objects covered by the interval from the begin-bound of the dominating object to the end-bound of object P. The covered objects are merged into a template object. Repeat the above process for each dominating object.

Finally, we can merge together those template objects and the remaining objects. This is the main idea of the *string generation algorithm*. How to merge objects into a template object is described later in the *template object generation algorithm*.

The *string generation algorithm* is described in detail as follows.

Algorithm. String generation
Input: $O_1(B_1, E_1), O_2(B_2, E_2), O_3(B_3, E_3), \ldots, O_n(B_n, E_n)$
Output: a 2D Z u-string (or v-string)

1. Sort in non-decreasing order all the begin-bound points and end-bound points B_i, E_i, $i = 1, 2, \ldots, n$.
2. Group the same value points into a same-value-list. Form a same-value-list sequence.
3. Loop from step 4 to step 8 for each same-value-list according to the non-decreasing order.
4. If there is no end-bound in the list, process the next same-value-list.
5. Find the dominating object from the objects in the same end-bound list so that the corresponding begin-bound of the dominating object is the smallest of them. Compute the size of the dominating object.
6. If there exist no objects partly overlapping with the dominating object, find the objects covered by the dominating object (including the dominating object itself). Call the *template object generation algorithm* with the covered objects as the input parameter.
7. If there exist objects partly overlapping with the dominating object, choose among them the object with the smallest end-bound. Let the chosen object be P. Perform the following three phrases.
   (a) Find the objects covered by object P (including object P itself). Then call the *template object generation algorithm* with the covered objects as the input parameter. Let the returned template object be T_1.
   (b) Find the objects covered by the dominating object but not by object P (including the dominating object itself). Then call the *template object generation algorithm* with the covered objects as the input parameter. Let the returned template object be T_2.
   (c) Merge T_1 and T_2 into a new template object by separators "(" and ")" with operator "/". The distance associated with operator "/" is equal to the end-bound of object T_2 minus the begin-bound of the object T_1. The begin-bound of the template object is equal to the begin-bound of object T_2. The end-bound of the template object is equal to the end-bound of object T_1.
8. Collect the begin-bounds and end-bounds of the objects into the same-value lists.
9. Call the *template object generation algorithm* with all the remaining objects as the input parameter. Output the representation of the final object. This is the u- (or v-)string.

Next, we define some terminology used in the *template object generation algorithm*. A former object of object O is an object with smaller begin-bound than that of object O or an object with equal begin-bound and bigger end-bound than that of object O. The nearest former object is the former object with the biggest begin-bound. If the number of such objects is more than one, choose one with the smallest end-bound as the nearest former object.

Given a list of objects, there exists an object, Q, that is not any objects' former objects. The begin-bound of object Q should be the largest among those of the objects in the list. If the number of objects with the largest begin-bound is more than one, object Q should be the object with the

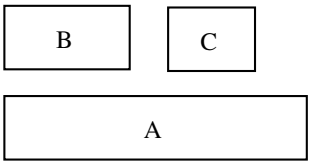


Fig. 3. Nearest former objects.

smallest end-bound. If the number of such objects is more than one, it means that they have the same begin-bound and end-bound and they can be merged together by operator "=" into a template object. Let Q be the newly merged template object. Hence, for a list of objects, there exists a unique object, Q, that is not any objects' former objects.

Let us consider the example as shown in Fig. 3. Object A is the nearest former object of object B. Object B is the nearest former object of object C. Object C is the object that is not any objects' former objects since the begin-bound of the x-projection of object C is the largest among objects A, B, and C.

Actually, we can use the relationship of the nearest former object to decide the order of merging objects into a template object. In the example shown in Fig. 3, objects B and C are first merged into a template object $(B < C)$ since object C is not any objects' former objects and object B is its nearest former object. Then, objects $(B < C)$ and A are merged into a template object $(A[(B < C))$ since object $(B < C)$ is not any objects' former objects and object A is its nearest former object, where the metric measurements are omitted. How to merge objects into a template object is described in detail in the *template object generation algorithm*.

Algorithm. Template object generation
Input: a list of objects
Output: a template object

1. Repeat steps 2–5 until there is only one object in the list.
2. For the objects having the same begin-bound and end-bound, they are chained by operator "=" in alphabetical order and form a template object. If there is only one object in the list, exit the repeat-loop.
3. For each object, find its nearest former object.
4. Let Q be the object that is not any objects' former objects and N be the nearest former object of object Q. Use the following phases to merge objects Q and N.
   (a) If $\mathrm{Begin}(N) = \mathrm{Begin}(Q)$ and $\mathrm{End}(N) > \mathrm{End}(Q)$, use operator "[" to merge objects N and Q. Go to step 5.
   (b) If $\mathrm{End}(N) = \mathrm{End}(Q)$ and $\mathrm{Begin}(N) < \mathrm{Begin}(Q)$, use operator "]" to merge objects N and Q. Go to step 5.
   (c) If $\mathrm{End}(N) > \mathrm{Begin}(Q)$ and $\mathrm{Begin}(N) < \mathrm{Begin}(Q)$, use operator "/" to merge objects N and Q. The distance associated with operator "/" is equal to $\mathrm{End}(N) - \mathrm{Begin}(Q)$. Go to step 5.
   (d) If $\mathrm{End}(N) < \mathrm{Begin}(Q)$, use operator "<" to merge objects N and Q. The distance associated with operator "<" is equal to $\mathrm{Begin}(Q) - \mathrm{End}(N)$. Go to step 5.
   (e) If $\mathrm{End}(N) = \mathrm{Begin}(Q)$, use operator "|" to merge objects N and Q. Go to step 5.
   (f) If $\mathrm{End}(N) > \mathrm{End}(Q)$, use operator "%" to merge objects N and Q. The distance associated with operator "%" is equal to $\mathrm{Begin}(Q) - \mathrm{Begin}(N)$.
5. If either one of objects Q and N is not a template object, compute the size of the object. Then objects Q and N are merged into a new template object by separators "(" and ")" with the appropriate operator found in step 4.

To see how the *string generation algorithm* works, let us consider the example as shown in Fig. 4. The locations of the objects in the image can be represented with the begin-bounds and end-bounds of their x-projections and form an object list as follows.

$A(0,4),\ B(1,2),\ C(2,4),\ D(5,9),\ E(6,8),\ F(8,10),\ G(9,10).$

Then we demonstrate how we apply the *string generation algorithm* to the above object list in order to obtain a 2D Z u-string.

By scanning the input object list from left to right, we shall find that the first dominating object is B. Since there is only one object covered by B (B

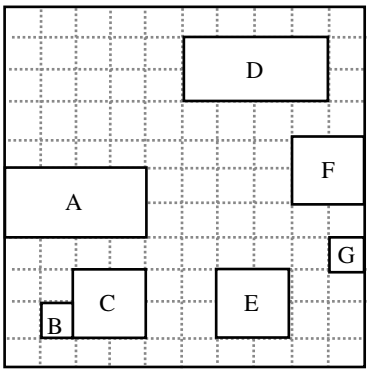


Fig. 4. The example of string generation.

itself) in step 6 of the *string generation algorithm*, process the next same-value list. The second dominating object is A since objects A and C have the same end-bound and the begin-bound of object A is smaller. In step 6 of the *string generation algorithm*, there exist no objects partly overlapping with object A. The objects covered by object A are objects A, B, and C. Call the *template object generation algorithm* with objects A, B, and C as the input parameter.

In the first repeat-loop of the *template object generation algorithm*, steps 2, 3, 4(e), and 5 are executed since object C is not any objects' former objects and object B is its nearest former object. It generates a template object $(B_1 \,|\, C_2)$. In the second repeat-loop of the *template object generation algorithm*, steps 2, 3, 4(b), and 5 are executed since object $(B_1 \,|\, C_2)$ is not any objects' former objects and object A is its nearest former object. It generates a template object $(A_4 \,]\, (B_1 \,|\, C_2))$.

The third dominating object is E. Since there is only one object covered by E (E itself) in step 6 of the *string generation algorithm*, process the next same-value list. The fourth dominating object is D. In step 7 of the *string generation algorithm*, the object partly overlapping with object D is object F. In step 7(a), the objects covered by object F are objects F and G. Call the *template object generation algorithm* with objects F and G as the input parameter. In the first repeat-loop of the *template object generation algorithm*, steps 2, 3, 4(b), and 5 are executed since object G is not any objects' former objects and object F is its nearest former object. It generates a template object $(F_2 \,]\, G_1)$. In step 7(b), the objects covered by object D but not by object F are objects D and E. Call the *template object generation algorithm* with objects D and E as the input parameter. In the first repeat-loop of the *template object generation algorithm*, steps 2, 3, 4(f), and 5 are executed since object E is not any objects' former objects and object D is its nearest former object. It generates a template object $(D_4 \,\%_1\, E_2)$. In step 7(c), both template objects $(D_4 \,\%_1\, E_2)$ and $(F_2 \,]\, G_1)$ are merged into a new template object $((D_4 \,\%_1\, E_2) \,/_1\, (F_2 \,]\, G_1))$.

At this point, because all objects are merged into template objects, step 9 of the *string generation algorithm* is executed. In this case, we call the *template object generation algorithm* with all the remaining objects (objects $(A_4 \,]\, (B_1 \,|\, C_2))$ and $((D_4 \,\%_1\, E_2) \,/_1\, (F_2 \,]\, G_1))$) as the input parameter. In the first repeat-loop of the *template object generation algorithm*, steps 2, 3, 4(d), and 5 are executed. It generates a template object $((A_4 \,]\, (B_1 \,|\, C_2)) <_1 ((D_4 \,\%_1\, E_2) \,/_1\, (F_2 \,]\, G_1)))$.

Finally, all the objects are merged together as a template object. So, the corresponding u-string of the image shown in Fig. 4 can be represented as $((A_4 \,]\, (B_1 \,|\, C_2)) <_1 ((D_4 \,\%_1\, E_2) \,/_1\, (F_2 \,]\, G_1)))$.

Lemma 1. *For an input list containing n objects, the length of a* 2D Z u- (*or* v-)*string generated by the string generation algorithm is bounded to* $O(n)$.

Proof. Let us first ignore the metric measurements.

If the input list contains just one object, that is $n = 1$, the generated 2D Z u- (or v-)string just contains the object itself. So, the length of the string is bounded to $O(1)$.

If the input list contains more than one object, let L_n be the length of the generated 2D Z u- (or v-)string. Let Q be an object that is not any objects' former objects and N be Q's nearest former object. Objects Q and N can be merged into a template object by one of the following operators: "[", "]", "/", "<", "|", and "%". The length of the template object is equal to the summation of the lengths of symbols: "(", Q, "op", N, and ")", where "op" is one of the following operators: "[", "]", "/", "<", "|", and "%". So, the length of the template object is a constant c. Since objects Q and N are merged

into one template object, there are $n-1$ objects to be processed. So we have

$$\begin{aligned} L_n &= L_{n-1} + c \\ &= L_{n-2} + c * 2 \\ &\vdots \\ &= L_1 + c * (n-1) \\ &= \mathrm{O}(1) + c * (n-1) \\ &= c * n - c + \mathrm{O}(1) \end{aligned}$$

So, the length of the generated 2D Z u- (or v-)string is bounded to $\mathrm{O}(n)$.

Since there are no cuttings between objects, there are n objects and $n-1$ operators in the generated 2D Z u- (or v-)string. For each object in the string, it contains a metric measurement. For each operator in the string, it contains at most one metric measurement. Hence, the number of metric measurements is bounded to $\mathrm{O}(n)$. Therefore, the length of the generated 2D Z u- (or v-)string is bounded to $\mathrm{O}(n)$. □

Theorem 1. *For an input list containing n objects, the length of a* 2D Z*-string generated by the string generation algorithm is bounded to* $\mathrm{O}(n)$.

Proof. By Lemma 1, we know that the u-string generated by the *string generation algorithm* is bounded to $\mathrm{O}(n)$. We also know that the v-string generated by the *string generation algorithm* is bounded to $\mathrm{O}(n)$. Therefore, for an input list containing n objects, the length of a 2D Z-string generated by the *string generation algorithm* is bounded to $\mathrm{O}(n)$. □

4. Image reconstruction algorithm

This section presents the *image reconstruction algorithm* which converts a 2D Z-string into a symbolic picture for visualization and browsing of image databases. The *image reconstruction algorithm* processes a 2D Z u-string (or v-string) to construct the locations, sizes, and distances for the objects in a symbolic picture.

In the *image reconstruction algorithm*, we introduce the notations about a string object S and an image object O used in our algorithm. Suppose that a given string (u- or v- string) consists of n elements, each of which may be a string object or operator. Each string object has the metric measurement (size) associated with it. Operators "<", "%", and "/" also have the metric measurement (distance) associated with them. For each element, W, of string objects, W.sym represents the string symbol and W.size represents the size associated with it. For each element, W, of operators "%", "<", or "/", W.sym represents the operator symbol and W.size represents the distance associated with it. For the other operators, " = ", "|", "[", and "]", the size fields associated with them are set to zero. Similarly, an image object O contains three fields: O.sym, O.size, and O.location. O.sym, O.size, and O.location represent the symbol, size, and the location of object O, respectively.

Assume that there are n elements in a given 2D Z u-string (or v-string) and m string objects in the n elements. The *image reconstruction algorithm* converts a given 2D Z u-string (or v-string) into a sequence of m image objects. After both sequences of image objects are derived from the given u- and v-strings, we have finished the image reconstruction. The *image reconstruction algorithm* is described in detail as follows.

Algorithm. Image reconstruction
Input: a 2D Z u-string (or v-string) with n elements: string $= (W_1, W_2, \ldots, W_n)$
Output: a list of image objects: ObjectList $= (O_1, O_2, \ldots, O_m)$

```
1.  Loc = 0; ObjectList = null; Stack = null; i = 1;
    j = 0; /* Initialization */
2.  MoreOperators = False;
3.  while (more elements in the 2D Z u- (or v-)
    string) /* process 2D Z u- or v-strings */
4.    while (MoreOperators)
5.      i = i + 1; /* next operator */
6.      MoreOperators = False;
7.      case W_i.sym
8.        "%": Loc = Loc + W_i.size; i = i + 1;
9.        "<": Loc = Loc + PreviousObjectSize +
                W_i.size; i = i + 1;
10.       "/": Loc = Loc + PreviousObjectSize -
                W_i.size; i = i + 1;
```

```
11.          "|": Loc = Loc + PreviousObjectSize;
                  i = i + 1;
12.          "]": If W_{i+1}.sym ≠ "(" then Template-
                     Size = W_{i+1}.size;
13.               else TemplateSize = GetTemplate-
                       Size (i + 1, string);
14.               end-if
15.               Loc = Loc + PreviousObjectSize
                        – TemplateSize;
16.               i = i + 1;
17.          "=" or "[": i = i + 1;
18.          ")": Pop a template object W from
                  Stack;
19.               Loc = W.beginBound;
20.               PreviousObjectSize = W.size;
21.               MoreOperators = True;
22.       end-case
23.    end-while
24.    while (W_i.sym = "(")
25.       Create the associated template object W;
26.       W.beginBound = Loc;
27.       W.size = GetTemplateSize (i, string);
28.       Push the template object W into Stack;
29.       i = i + 1;
30.    end-while
31.    if W_i is a string object then
32.       j = j + 1;
33.       Create a new object O_j so that
34.          O_j.sym = W_i.sym;
35.          O_j.size = W_i.size;
36.          O_j.beginBound = Loc;
37.       Append object O_j to ObjectList.
38.    end-if
39.    PreviousObjectSize = W_i.size;
40.    MoreOperators = True;
41. end-while
42. Output the ObjectList.
```

The function GetTemplateSize calculates the size of the template object at the next level. The size of the template object is equal to the summation of:

1. the size of the first element after "(",
2. the size of the element after operator "|",
3. the size of the element after operator "<",
4. the distance associated with operator "<",
5. the size of the element after operator "/",
6. the negative distance associated with operator "/".

Notice that it is not necessary to calculate the sizes of template objects at third or lower levels.

5. Performance analysis

To compare our *string generation and image reconstruction algorithms* with those of the 2D C^+-string, we perform a series of experiments, which is made on the synthesized images. There are two cost factors dominating the performance of the *string generation and image reconstruction algorithms*: the number of images and the number of objects in an image. We freely set the values of the two cost factors in the synthesized images. All the algorithms are implemented on an IBM compatible personal computer of Pentium III-800 with Windows 2000.

First, we compare our *string generation and image reconstruction algorithms* with those of 2D C^+-string by using synthesized images. The execution cost of every experiment is measured by the average elapsed time of image processing. We generate the image index for 1000 images. For each image, we assign 25 objects. Based on these synthesized images, we perform four experiments. The experimental results are shown as follows.

Fig. 5 illustrates the execution time versus the number of images for the *string generation and image reconstruction algorithms*. Each image in these two experiments contains 25 objects. The execution time grows linearly as the number of images increases. The 2D C^+-string approach spends about 50–80% more time than the 2D Z-string approach to generate a string or to reconstruct an image.

Fig. 6 illustrates the execution time versus the number of objects in an image for the *string generation and image reconstruction algorithms*. In these two experiments, we run 1000 images for each case. The execution time is averaged over the execution time for each image. The execution time grows linearly as the number of objects in an image increases for the 2D Z-string approach. The 2D C^+-string approach spends about 50–100%

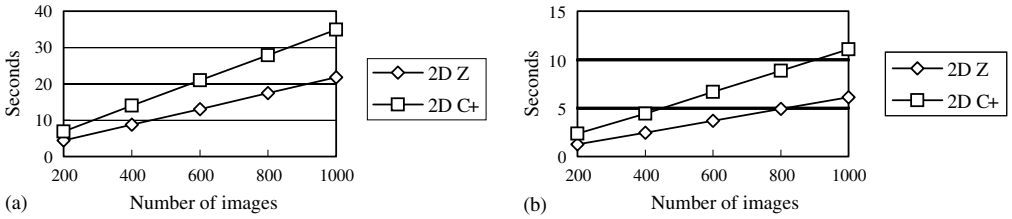


Fig. 5. The execution time vs. the number of images: (a) string generation algorithm and (b) image reconstruction algorithm.

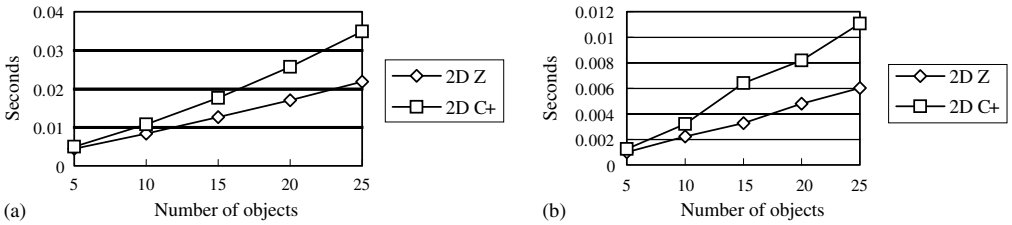


Fig. 6. The execution time vs. the number of objects in an image: (a) string generation algorithm and (b) image reconstruction algorithm.

more time than the 2D Z-string approach to generate a string or to reconstruct an image.

Fig. 7 illustrates the string length vs. the number of objects in an image for the *string generation algorithm*. In this experiment, we run 1000 images for each case. The string length is averaged over the string length for each image. The spatial operators and the parentheses occupy one byte of storage. The object symbols and metric measurements occupy four bytes of storage. The string length grows linearly as the number of objects in an image increases for the 2D Z-string approach. However, the string length grows sharply as the number of objects in an image increases for the 2D C^+-string approach.

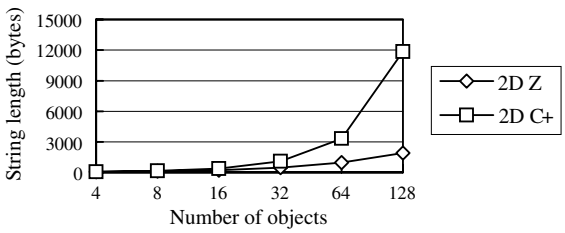


Fig. 7. The string length vs. the number of objects in an image.

To compare the efficiency of similarity retrieval for both approaches, we perform a query in which we retrieve the similar images with the equal spatial relations in the x and y dimensions. In this experiment, we run 100 queries, each of which contains 10 objects. The execution time is averaged over the execution time for each query. Fig. 8(a)

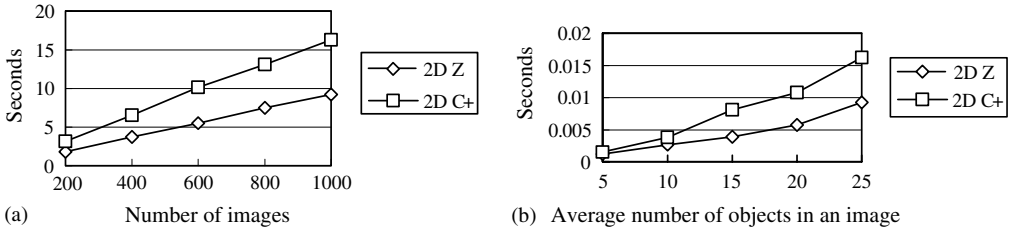


Fig. 8. Similarity retrieval: (a) the execution time vs. number of images and (b) the execution time vs. number of objects in an image.

presents the execution time vs. the number of images. The execution time grows linearly as the number of images increases. Fig. 8(b) illustrates the execution time vs. the average number of objects in an image. The execution time grows as the average number of objects in an image increases. Generally speaking, the 2D C^+-string approach spends about 50–100% more time than the 2D Z-string approach in performing similarity retrieval.

In summary, the 2D C^+-string approach spends about 50–100% more time than the 2D Z-string approach to generate a string, to reconstruct an image, or to perform similarity retrieval since the string lengths generated by the 2D Z-string and 2D C^+-string approaches are bounded by $O(n)$ and $O(n^2)$, respectively. The longer the generated string is, the more execution time is required.

6. Concluding remarks

The approach of 2D strings opens a new area for iconic picture indexing and retrieval. There are many follow-up indexing methods based on the concept of 2D string including 2D G-string proposed by Chang et al. (1988), 2D C-string proposed by Lee and Hsu (1990, 1991, 1992), 2D C^+-string proposed by Huang and Jean (1994), and 2D RS-string proposed by Huang and Jean (1996). However, all previously proposed methods are not economical and efficient in terms of storage space and execution time due to their cutting mechanisms. The order of the total number of partitioned subparts is $O(n^2)$ in the worst case, where \n is the number of objects in the image.

To overcome this problem, we propose a new spatial knowledge representation 2D Z-string. Since there are no cuttings between objects in the knowledge structure of 2D Z-string, the integrity of objects is preserved and the string length is bounded to $O(n)$. The shorter the generated string is, the less execution time is required for string generation and image reconstruction.

Since the 2D Z-string is extended from the 2D C-string and 2D C^+-string, the spatial inference and similarity retrieval algorithms proposed in the 2D C-string or 2D C^+-string can be applied directly to the 2D Z-string to retrieve the similar images from a database. The 2D Z-string representation can capture the precise and compact spatial relations between objects. Our new knowledge structure can be easily applied to an intelligent image database system to reason about spatial relations between the objects in an image.

Acknowledgements

This research was supported in part by Center for Information and Electronics Technologies (CIET), National Taiwan University and the National Science Council of Republic of China under Grant No. NSC-91-2213-E-002-092.

References

Chang, S.K., Jungert, E., 1986. A spatial knowledge structure for image information systems using symbolic projections. In: Proc. Fall Joint Computer Conf., pp. 79–86.

Chang, S.K., Shi, Q.Y., Yan, C.W., 1987. Iconic indexing by 2D strings. IEEE Trans. Pattern Anal. Mach. Intell. 9, 413–429.

Chang, S.K., Jungert, E., Li, Y., 1988. Representation and retrieval of symbolic pictures using generalized 2D strings. Technical Report, University of Pittsburgh.

Chang, Y.I., Ann, H.Y., Yeh, W.H., 2000. A unique-ID-based matrix strategy for efficient iconic indexing of symbolic pictures. Pattern Recognition 33, 1263–1276.

Chang, Y.I., Yang, B.Y., Yeh, W.H., 2001. A generalized prime-number-based matrix strategy for efficient iconic indexing of symbolic pictures. Pattern Recognition Lett. 22, 657–666.

Chang, Y.I., Yang, B.Y., Yeh, W.H., 2003. A bit-pattern-based matrix strategy for efficient iconic indexing of symbolic pictures. Pattern Recognition Lett. 24, 537–545.

Jungert, E., 1988. Extended symbolic projections as a knowledge structure for spatial reasoning. In: Proc. 4th BPRA Conf. on Pattern Recognition, pp. 343–351.

Jungert, E., Chang, S.K., 1989. An algebra for symbolic image manipulation and transformation. Visual Database Systems. Elsevier Science Publishers BV, North-Holland.

Huang, P.W., Jean, Y.R., 1994. Using 2D C^+-string as spatial knowledge representation for image database systems. Pattern Recognition 27, 1249–1257.

Huang, P.W., Jean, Y.R., 1996. Spatial reasoning and similarity retrieval for image database systems based on RS-strings. Pattern Recognition 29, 2103–2114.

Lee, S.Y., Hsu, F.J., 1990. 2D C-string: A new spatial knowledge representation for image database systems. Pattern Recognition 23, 1077–1087.

Lee, S.Y., Hsu, F.J., 1991. Picture algebra for spatial reasoning of iconic images represented in 2D C-string. Pattern Recognition Lett. 12, 425–435.

Lee, S.Y., Hsu, F.J., 1992. Spatial reasoning and similarity retrieval of images using 2D C-string knowledge representation. Pattern Recognition 25, 305–318.

Petraglia, G., Sebillo, M., Tucci, M., Tortora, G., 2001. Virtual images for similarity retrieval in image databases. IEEE Trans. Knowledge Data Eng. 13 (6), 951–967.