An Efficient Iconic Indexing Strategy for Image Rotation and Reflection in Image Databases [1]

Wei-Horng Yeh and Ye-In Chang

Dept. of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan
Republic of China
{E-mail: yehwh@mail.cse.nsysu.edu.tw}
{Tel: 886-7-5254350}
{Fax: 886-7-5254301}

Abstract

Spatial relationships are important issues for similarity-based retrieval in many image database applications. With the popularity of digital cameras and the related image processing software, a sequence of images are often rotated or flipped. That is, those images are transformed in the rotation orientation or the reflection direction. However, many iconic indexing strategies based on symbolic projection are sensitive to rotation or reflection. Therefore, these strategies may miss the qualified images, when the query is issued in the orientation different from the orientation of the database images. To solve this problem, some researchers proposed a function to map the spatial relationship to its transformed one. However, this mapping consists of several conditional statements, which is time-consuming. Thus, in this paper, we propose an efficient iconic indexing strategy, in which we carefully assign a unique bit pattern to each spatial relationship and record the spatial information based on the bit patterns in a matrix. Without generating the rotated or flipped image, we can directly derive the index of the rotated or flipped image from the index of the original one by bit operations and matrix manipulation. In our performance study, we analyze the time complexity of our proposed strategy and show the efficiency of our proposed strategy according to the simulation results. Moreover, we implement a prototype to validate our proposed strategy.

(*Keywords*: iconic indexing, image databases, image rotation and reflection, query by image content, similarity retrieval)

[1]This research was supported in part by the National Science Council of Republic of China under Grant No. NSC-87-2213-E-110-014.

1 Introduction

High dimensional digital data in image database systems are popular in the image analysis, computer graphics, pattern recognition, geographic information system and several industrial domains [20]. Content-based retrieval plays an important role in multimedia database applications [2, 10, 16, 24, 25]. It can be classified into two fields: one includes the color, texture, and shape features, the other includes the spatial relationships between objects. Retrieval by Spatial Similarity (RSS) is used to process a number of queries that is based on spatial relationships among the domain objects. The goal of RSS is to retrieve database images that satisfy the spatial relationships specified in the query. Generally, an image database system stores images with the corresponding image indexing techniques concerning querying the database. An example of the query could be "find all pictures showing a house to the right of a river." Thus, the spatial information must be preserved by the index data structure. Methods for RSS can be roughly divided into symbolic projection [7, 17, 18], graph-matching [11, 26], and geometric [13, 29] categories. Since the symbolic projection can simplify the complexity of describing the positions of the objects in an image, in this paper, we focus on the symbolic projection category.

In the 3D animation field [15, 27], the sequence of images performed by computers are often operated by a number of basic transformations, *e.g.*, rotation and reflection. Figure 1-(a) is the original image. Figure 1-(b) to Figure 1-(f) show the results of five different basic transformations applied to the original one. Similarly, we often have to rotate or flip images in our real life. For example, when we scan a picture, we may get the image in the reverse orientation. The image processing software can employ the 180-degree rotation for us.

As in robotic scenes and virtual reality applications, they require to solve queries like this: "find those images that are satisfied with a given pattern even if it shows in a rotation orientation." However, indexing methods based on symbolic projection are sensitive to rotation or reflection. If we want to retrieve an image which is rotated and stored in the database and we have only the index that represent the image before being rotated, we must have the new index of the rotated image such that we can retrieve the image. There are two approaches to solve this problem. The first approach is that we reconstruct the image from the original index, rotate the image and construct the index of the rotated image as

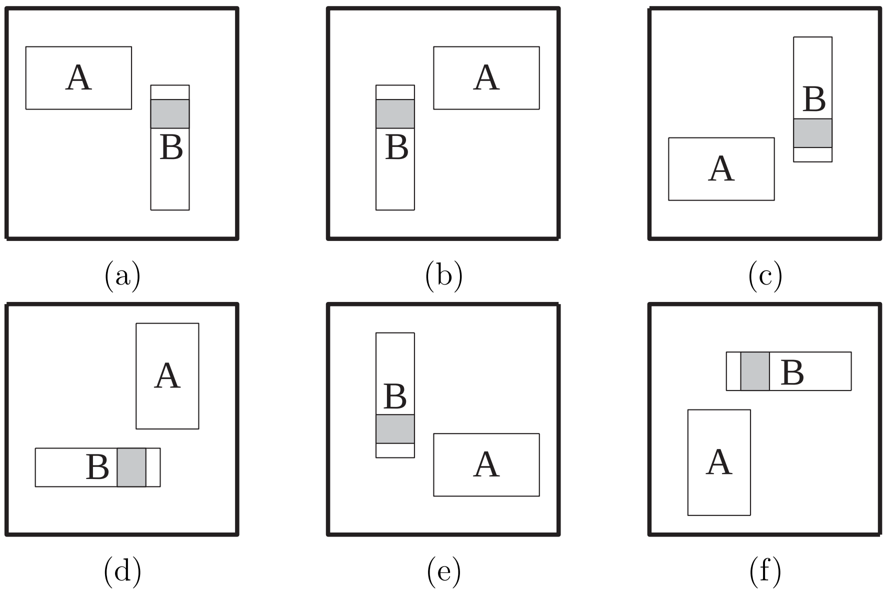


Figure 1: Image transformation: (a) the original image; (b) flipped horizontally; (c) flipped vertically; (d) rotated by 90°; (e) rotated by 180°; (f) rotated by 270°.

shown in Figure 2-(a). In order not to miss the qualified database images, we need to do steps 2 and 3 five times to derive the five indexes, *i.e.*, the indexes of the image rotated by 90, 180, and 270 degrees, and the indexes of the image flipped horizontally and vertically. Figure 2-(b) shows the second approach, in which we find a corresponding strategy such that the new index of the rotated image can be constructed by the original index. The first approach is time-consuming as compared to the second approach. Therefore, finding a good strategy such that we can efficiently get the new index of the adjusted image from the original index is important.

Petrakis [23] described that one of classes of the spatial image content representation and matching is *symbolic projection*. However, strategies based on symbolic projection, *e.g.*, [6, 14], cannot recognize similarity between two indexes corresponding to an image and one of its possible transformations, *e.g.*, rotation and reflection. To solve this problem, Nabil *et al.* [21] and Petraglia *et al.* [22] proposed a similar mapping for the rotation and reflection of the spatial relationships. However, the process of the index mapping is time-consuming. Thus, in this paper, we classify the mapping into three cases and carefully assign a 16-bit unique bit pattern to each spatial relationship. Based on the assignment, we

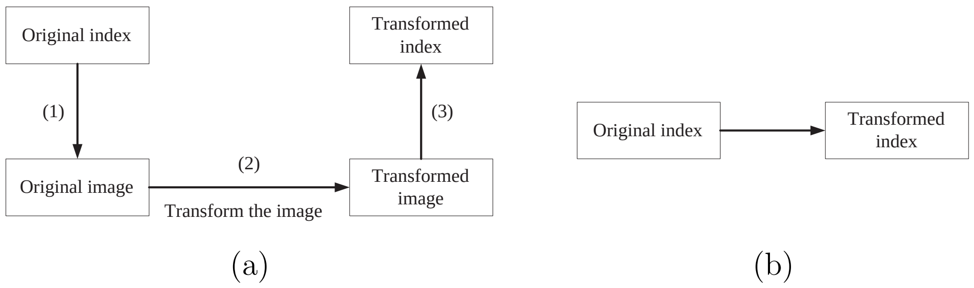


Figure 2: The process for obtaining the rotated or flipped index: (a) the original three steps; (b) the second approach.

can easily do the mapping with our proposed bit operation, *intra-exchange*. Moreover, we propose an efficient iconic index strategy, called *Unique Bit Pattern* matrix strategy (UBP matrix strategy) to record the spatial information. In this way, when doing similarity retrieval, we do not need to reconstruct the original image from the UBP matrix in order to obtain the indexes of the rotated and flipped image. Conversely, we can directly derive the index of the rotated or flipped image from the index of the original one through bit operations and the matrix manipulation. Thus, our proposed strategy can do similarity retrieval without missing the qualified database images. From our performance study, we show that our strategy outperforms those mapping strategies [21, 22] based on different number of objects in an image. The percentage of improvement is between 13.64% and 53.23%. Moreover, we implement a prototype to validate our proposed strategy.

The rest of this paper is organized as follows. Section 2 gives a brief description about some related work and image rotation/reflection. Section 3 presents our proposed strategy in detail. In Section 4, we make a simulation study to show that our proposed strategy is efficient, and present a prototype system. Finally, concluding remarks are made in Section 5.

2 Background

To reduce the complexity of constructing iconic indexes, each object in an image is abstracted as a Minimum Boundary Rectangle (MBR) to represent the size and position of the object. Table 1 shows the meaning of the spatial operators proposed by Lee and Hsu

Table 1: Definitions of Lee *et al.*'s spatial operators

Notation	Condition	Meaning
$A < B$	$\text{end}(A) < \text{begin}(B)$	A disjoins B
$A\|B$	$\text{end}(A) = \text{begin}(B)$	A is edge to edge with B
A/B	$\text{begin}(A) < \text{begin}(B) < \text{end}(A) < \text{end}(B)$	A is partly overlapping with B
$A]B$	$\text{begin}(A) < \text{begin}(B)$ $\text{end}(A) = \text{end}(B)$	A contains B and they have the same end bound
$A[B$	$\text{begin}(A) = \text{begin}(B)$ $\text{end}(A) > \text{end}(B)$	A contains B and they have the same begin bound
$A\%B$	$\text{begin}(A) < \text{begin}(B)$ $\text{end}(A) > \text{end}(B)$	A contains B and they do not have the same bound
$A = B$	$\text{begin}(A) = \text{begin}(B)$ $\text{end}(A) = \text{end}(B)$	A is at the same position as B

[18], where the notation "begin(A)" and "end(A)" denotes the beginning and ending points of the object A, respectively. Then, Chang and Lee [4] extended those 7 spatial operators to 13 ones by adopting Allen's 13 types of interval relationships [1] and introduced the inverse operators. Figure 3 shows the spatial operators and the inverse ones. Let us denote the white rectangle by A and the gray rectangle by B. If the spatial relationship between A and B is "$B < A$" as shown in the second row and the first column in Figure 3, then we could use the inverse operator to represent the spatial relationship as "$A <^* B$". According to the combination of those 13 spatial relationships in x- and y-axes, there are 169 spatial relationships in 2D space. They can be classified into five spatial categories, *disjoin, join, contain, belong,* and *partial overlapping.* The criterion for the spatial categories is the intersection area between each two objects. Figure 4 shows some examples of those five spatial categories.

Chang *et al.* [8] proposed an index structure called *unique-ID-based matrix* (UID matrix) for symbolic pictures, in which each spatial relationship between any two objects is assigned to a unique identifier and is recorded in a matrix. Table 2 shows the unique identifiers for those 13 spatial relationships. The entry m_{ij} in a UID matrix is the spatial relationship between objects i and j in x-axis when $i > j$. Otherwise, m_{ij} is the spatial relationship in y-axis when $i < j$. For example, the related UID matrix M for the symbolic image as

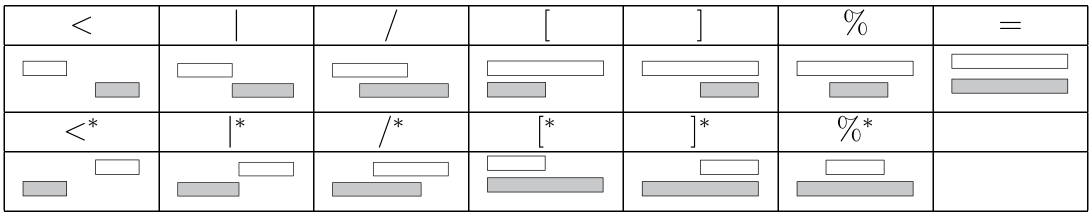


Figure 3: 13 types of spatial operators in one dimension (horizontal projection)

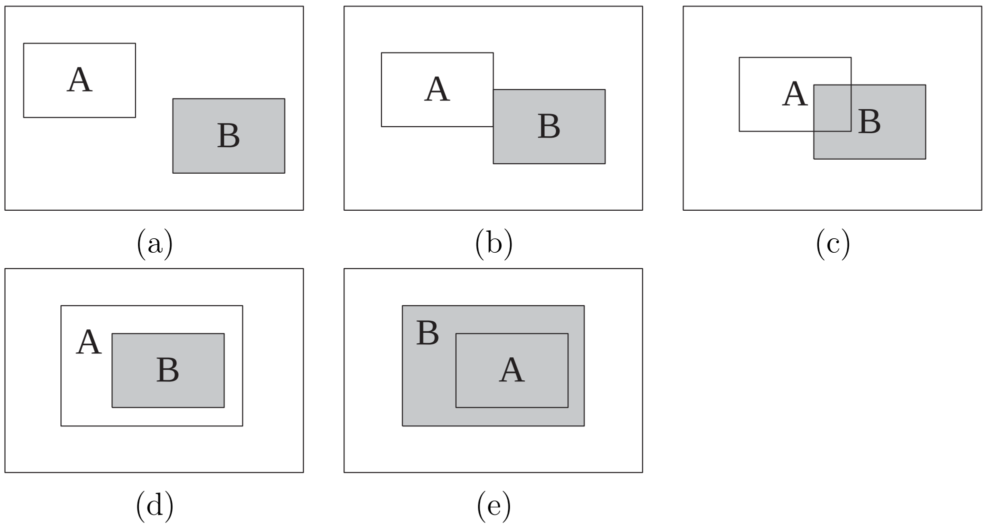


Figure 4: Examples of spatial categories: (a) disjoin; (b) join; (c) partial overlapping; (d) contain; (e) belong.

Table 2: Unique identifiers

operator	$<$	$<^*$	$\mid$	$\mid^*$	$/$	$/^*$	$]$	$[$	$\%$	$=$	$]^*$	$[^*$	$\%^*$
UID	1	2	3	4	5	6	7	8	9	10	11	12	13

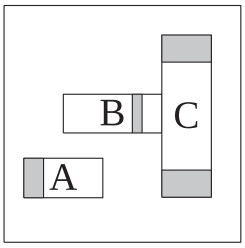


Figure 5: An example of a symbolic image presented by MBRs

shown in Figure 5 is as follows.

$$M = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 1 & 12 \\ 5 & 0 & 13 \\ 1 & 3 & 0 \end{bmatrix} \end{array}$$

The lower left triangular area of the matrix M records the spatial relationships among the x-axis, and the upper right one records the spatial relationships among the y-axis. For example, the unique identifiers for the objects A and B among the x- and y-axes are 5 and 1, respectively. Thus, this means that the object B is above the object A.

To deal with the change of spatial relationships between objects in rotation and reflection, Nabil *et al.* [21] introduced the condition function as shown in Figure 6. In Figure 6, $g(h)$ is a spatial relationship, and $gi(hi)$ is the inverse of $g(h)$. On the other hand, Petraglia *et al.* [22] presented a mapping table as shown in Table 3 to deal with the linear transformation. For example, the transformed operator of the spatial operator "$<$" is "$<^*$". The operator "$]^*$" is the transformed operator of the spatial operator "$[^*$". Comparing the condition function with the mapping table, they obtain the same transformed operator.

3 The Unique Bit Pattern Matrix Strategy

In this section, first, we describe the rules of transformation of the spatial relationships among objects. Next, we introduce the concept of the transpose of a matrix and propose a

$$\xi(g) = \begin{cases} gi & \text{if } g \in \{<, |, /\} \\ g & \text{if } g \in \{\%, \%^*, =\} \\ h & \text{where } g = hi \text{ and } g \in \{<^*, |^*, /^*\} \\ k & \text{where } g \in \{[, [^*\} \text{ and } k \in \{],]^*\} \\ l & \text{where } g \in \{],]^*\} \text{ and } l \in \{[, [^*\} \end{cases}$$

Figure 6: The condition function for linear transformations

Table 3: Transformed operators

operator	$<$	$<^*$	$\|$	$\|^*$	$/$	$/^*$	$]$	$[$	$\%$	$=$	$]^*$	$[^*$	$\%^*$
transformed operator	$<^*$	$<$	$\|^*$	$\|$	$/^*$	$/$	$[$	$]$	$\%$	$=$	$[^*$	$]^*$	$\%^*$

bit operation. Then, we propose special bit patterns to represent the spatial relationships. Finally, we present our proposed index structure and the relevant algorithms.

3.1 Rules of Image Rotation and Reflection

From Table 3, we observe that the transformed operators can be classified into three cases as shown in Table 4. In Case 1, the related transformed operator is the same as the related inverse operator. For example, in Figure 7, "$<^*$" is both of the transformed operator and the inverse operator of the spatial operator "$<$". In Case 2, the related transformed operator is different from the related inverse operator. For example, in Figure 8, the transformed operator of the spatial operator "]" is "[", but the inverse operator is "$]^*$". In Case 3, the related transformed operators are themselves. For example, in Figure 9, the transformed operator of the spatial operator "%" is itself, *i.e.*, "%".

When we rotate an image in the clockwise direction as shown in Figure 10-(a), the spatial relationships between objects A and B in the x- and y-axes are mutually exchanged.

Table 4: Transformed operators divided into 3 cases

	Case 1			Case 2		Case 3		
operator	$<$	$\|$	$/$	$]$	$]^*$	$\%$	$=$	$\%^*$
transformed operator	$<^*$	$\|^*$	$/^*$	$[$	$[^*$	$\%$	$=$	$\%^*$

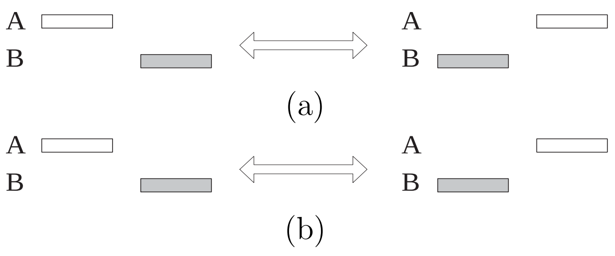


Figure 7: Example of Case 1: (a) $A < B$ *vs.* $A <^* B$ (transformed); (b) $A < B$ *vs.* $A <^* B$ (inverse).

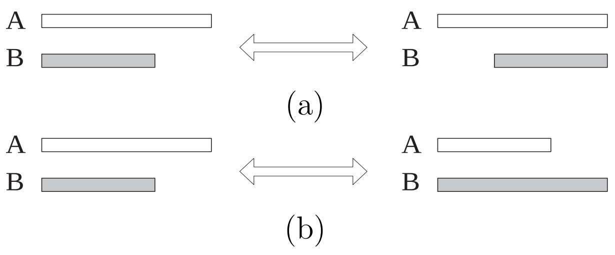


Figure 8: Example of Case 2: (a) $A[B$ *vs.* $A]B$ (transformed); (b) $A[B$ *vs* $A[^*B$ (inverse).

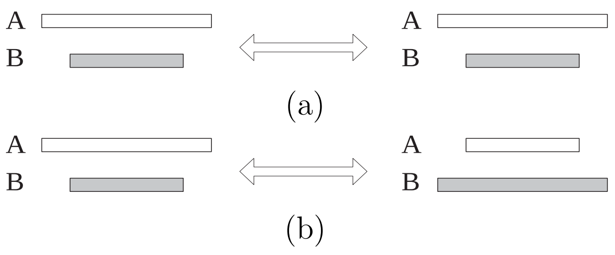
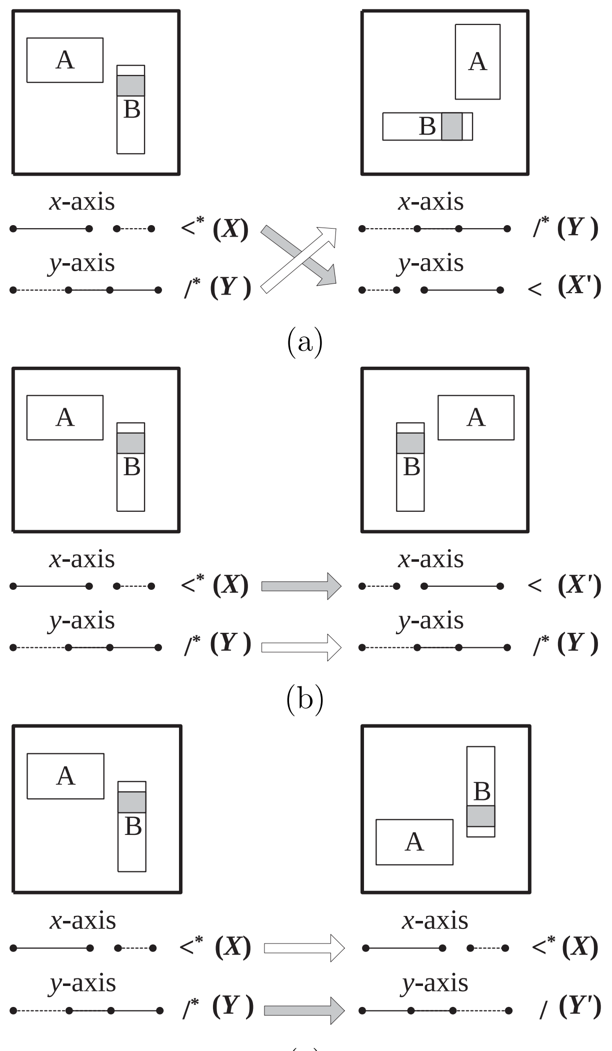


Figure 9: Example of Case 3: (a) $A\%B$ *vs.* $A\%B$ (transformed); (b) $A\%B$ *vs.* $A\%^*B$ (inverse).



X' (Y'): The transformed operator of X (Y)

Figure 10: Examples of rotation and reflection: (a) rotating 90° clockwise; (b) flipping horizontally; (c) flipping vertically.

Table 5: Rules of transformation: X' and Y' are the transformed operators related to X and Y, respectively.

	operator	
	x-axis	y-axis
functions	X	Y
Rotate 90°	Y	X'
Rotate 180°	X'	Y'
Rotate 270°	Y'	X
Flip horizontally	X'	Y
Flip vertically	X	Y'

Moreover, the beginning and the ending points of any object in the y-axis are mutually exchanged. Thus, the crossed arrows in Figure 10-(a) show the exchange of the spatial operators in the x- and y-axes. Moreover, the gray arrow shows the spatial operator, *i.e.*, "$A <^* B$", changes to the related transformed operator, *i.e.*, "$A < B$", in the rotated image. When we flip an image horizontally as shown in Figure 10-(b), the beginning and ending points of any object in the x-axis are mutually exchanged. Thus, the gray arrow in Figure 10-(b) shows the spatial operator, *i.e.*, "$A <^* B$", changes to the related transformed operator, *i.e.*, "$A < B$", in the horizontally flipped image. When we flip an image vertically as shown in Figure 10-(c), the beginning and ending points of any object in the y-axis are mutually exchanged. Thus, the gray arrow in Figure 10-(c) shows the spatial operator, *i.e.*, "$A/^*B$", changes to the related transformed operator, *i.e.*, "A/B", in the vertically flipped image. According to the above observation, we present rules for transformation (rotation and reflection) as shown in Table 5. It is obvious that rotating an image by 180° and 270° clockwise are equivalent to rotating the image by 90° clockwise twice and three times, respectively. Thus, it is trivial to derive the rules of rotating images by 180° and 270° clockwise.

3.2 The Matrix Manipulation and the Proposed Bit Operation

In Table 5, we observe that the spatial relationships in x- and y-axes will be mutually exchanged, when the image is rotated by 90° or 270° clockwise. Similar to the UID matrix

```
procedure Transpose(M) /* M is an $n \times n$ UBP matrix */
1:   for $i := 1$ to $n$ do
2:     for $j := i + 1$ to $n$ do
3:     begin
4:       $temp := M[i, j]$;
5:       $M[i, j] := M[j, i]$;
6:       $M[j, i] := temp$;
7:     end
8:   return $(M)$;
end procedure
```

Figure 11: Procedure *Transpose*

strategy [8], we will use a matrix, called *unique bit pattern* matrix (UBP matrix), to record the spatial relationships among objects. The entry m_{ij} in a UBP matrix is the spatial relationship between objects i and j in x-axis when $i > j$. Otherwise, m_{ij} is the spatial relationship in y-axis when $i < j$. Because we use a matrix to be as the index structure, we introduce the concept of the *transpose* of a matrix to deal with the mutual exchange of the spatial relationships in x- and y-axes. Definition 1 describes the meaning of the transpose of a matrix. Figure 11 shows the procedure to obtain the transpose of a matrix.

Definition 1. *The matrix B is the* ***transpose*** *of the matrix A, written $B = A^T$, if each entry b_{ij} in B is the same as the entry a_{ji} in A, and conversely [12].*

Suppose M is a UBP matrix, and N is the transpose of the matrix M, *i.e.*, $N = M^T$. According to the definition of the transpose of a matrix, m_{ij} in M is the same as n_{ji} in N. m_{ij} is the spatial relationship between the objects i and j in x-axis when $i > j$. Then, n_{ji} which is the same as m_{ij} is to be the spatial relationship between objects i and j in y-axis. This means that the spatial relationships between the two objects in x-axis in M are to be the spatial relationships in y-axis in N. As the same result, the spatial relationships in y-axis in M are to be the spatial relationships in x-axis in N. In this way, the meaning of the transpose of a UBP matrix is to mutually exchange the spatial relationships in x- and y-axes.

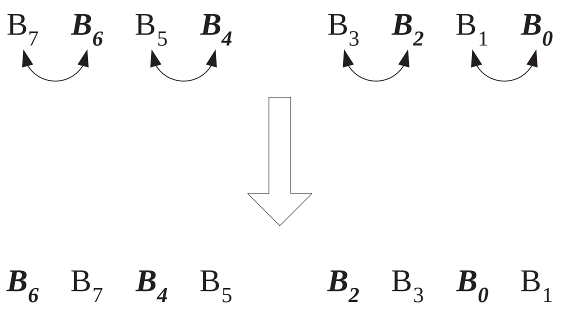

B_7 B_6 B_5 B_4 B_3 B_2 B_1 B_0
B_6 B_7 B_4 B_5 B_2 B_3 B_0 B_1

Figure 12: Intra-exchange of the odd and even bits

When doing the rotation or flip operations, we need to change the spatial relationships to their related transformed spatial relationships by following the rules shown in Table 5. Thus, we propose a bit operation and use bit patterns to present those 13 spatial relationships to make the change more efficient.

There are several bit operations, *e.g.*, *bit shift*, *and*, *or*, and *exclusive or*, *etc.* We propose a bit operation, called *intra-exchange*, as shown in Figure 12. The bits B_i and B_{i+1} are mutually exchanged, where i is an even number. Figure 13 shows the procedure to do the *intra-exchange* operation. W_1 and W_2 are the bit patterns, where the odd and even bits are set to 1, respectively.

We use an example as shown in Figure 14 to describe the steps of Procedure *Intra_Exchange*. First, we shift left the bit string "0000 1001" by 1 to obtain T_{left}, *i.e.*, "0001 0010". Second, we shift right the bit string "0000 1001" by 1 to obtain T_{right}, *i.e.*, "0000 0100". To extract the values of the odd bits in T_{left}, we do the *AND* operation with T_{left} and W_1 to obtain T_{odd}, *i.e.*, "0000 0010". Similarly, to extract the values of the even bits in T_{right}, we do the *AND* operation with T_{right} and W_2 to obtain T_{even}, *i.e.*, "0000 0100". Finally, we do the *OR* operation with T_{even} and T_{odd} to obtain the intra-exchanged bit string, *i.e.*, "0000 0110".

3.3 Unique Bit Patterns

Chang *et al.* [8] assigned 13 different numbers, *i.e.*, from 1 to 13, to those 13 spatial operators as shown in Table 2. We observe that those operators and their related transformed operators shown in Cases 1 and 2 in Table 3 are adjacent to each other. Although the

```
procedure Intra_Exchange(bit_string)
1:    $T_{left}$ := bit_string shift left by 1;
2:    $T_{right}$ := bit_string shift right by 1 ;
3:    $T_{odd} := T_{left}$ AND $W_1$; /* $W_1 = 101010\ldots10$, where $|W_1| = |bit\_string|$ */
4:    $T_{even} := T_{right}$ AND $W_2$; /* $W_2 = 010101\ldots01$, where $W_2 = \overline{W_1}$ */
5:    $result := T_{even}$ OR $T_{odd}$;
6:    return (result);
end procedure
```

Figure 13: Procedure *Intra_Exchange*

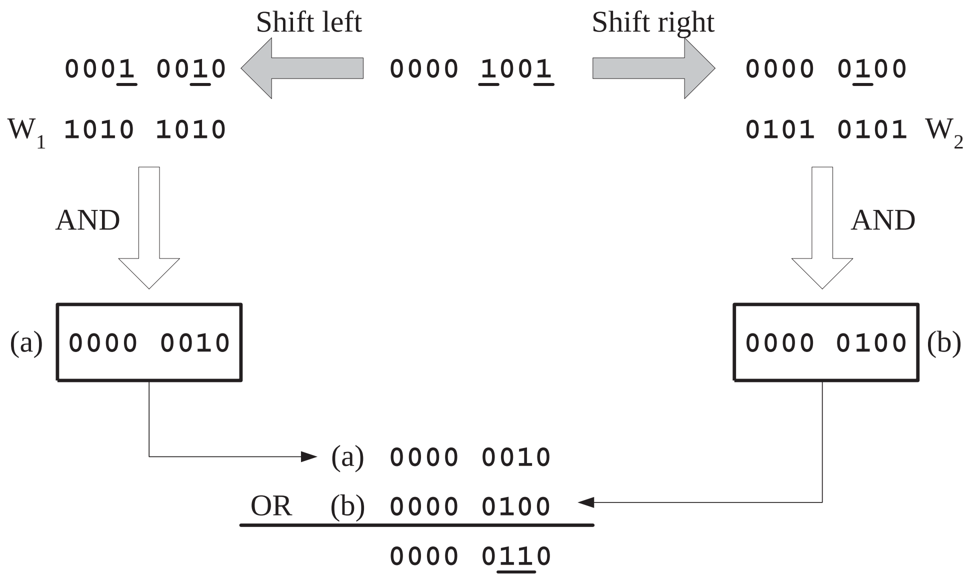


Figure 14: Example of applying Procedure *Intra_Exchange* to bit string 0000 1001

Table 6: New order of operators

	Case 1						Case 2		Case 3			Case 2	
operator	$<$	$<^*$	$\mid$	$\mid^*$	$/$	$/^*$	$]$	$[$	$\%$	$=$	$\%^*$	$]^*$	$[^*$
order	1	2	3	4	5	6	7	8	9	10	**11**	**12**	**13**

related transformed operators of those three operators shown in Case 3 in Table 3 are themselves, we change the order of the last three operators shown in Table 2 to make these three operators of Case 3, *i.e.*, "%", "=", "%*", be adjacent to each other. Table 6 shows the 13 spatial operators in the new order.

According to the order of the spatial operators shown in Table 6, We assign a unique 16-bit pattern, instead of a unique number, to each spatial operator as shown in Table 7. The second column shows the spatial operators, and the third column shows their corresponding bit patterns. The last column shows the decimal numbers, when we view the bit patterns as binary numbers. There is only 1 bit set to 1 in the bit patterns of those operators in Case 1 and 2. However, the bit patterns of those operators in Case 3 are assigned with two of 1's. That is why we need total $(10 \times 1 + 3 \times 2)$ bits to represent each spatial operator. In this assignment, we observe that the 1's in the bit patterns are adjacent to each other with the operator and the related transformed operator. For example, for the operator "$<$" and its related transformed operator "$<^*$", the bit which is set to 1 is in B_0 and B_1 bit, respectively, where B_0 is the right most bit of a bit pattern. In this way, by applying the *Intra_Exchange* procedure to the bit patterns of the spatial operators, the result is the bit pattern of the related transformed operators. For example, in Figure 15-(a), doing the *intra-exchange* operation on the bit pattern of the operator "$<$" results in the bit pattern of the transformed operator, "$<^*$". Figure 15-(b) shows the example of doing *intra-exchange* operation on the operators "]" and "[" which belong to Case 2. In Figure 15-(c), the transformed operator of "%" is itself, and doing the *intra-exchange* operation on the bit pattern of "%" also generates the bit pattern of itself.

Table 7: Bit patterns of operators

Case	Operator	Unique bit pattern	Decimal number
1	$<$	0000 0000 0000 000**1**	1
	$<^*$	0000 0000 0000 00**1**0	2
	$\vert$	0000 0000 0000 0**1**00	4
	$\vert^*$	0000 0000 0000 **1**000	8
	$/$	0000 0000 000**1** 0000	16
	$/^*$	0000 0000 00**1**0 0000	32
2	$]$	0000 0000 0**1**00 0000	64
	$[$	0000 0000 **1**000 0000	128
3	$\%$	0000 00**11** 0000 0000	768
	$=$	0000 **11**00 0000 0000	3072
	$\%^*$	00**11** 0000 0000 0000	12288
2	$]^*$	0**1**00 0000 0000 0000	16384
	$[^*$	**1**000 0000 0000 0000	32768

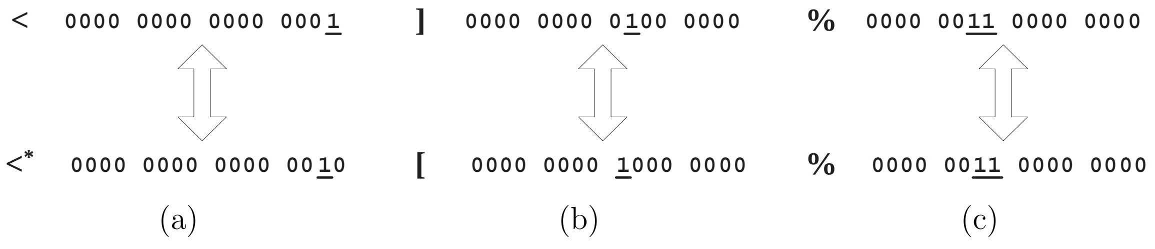
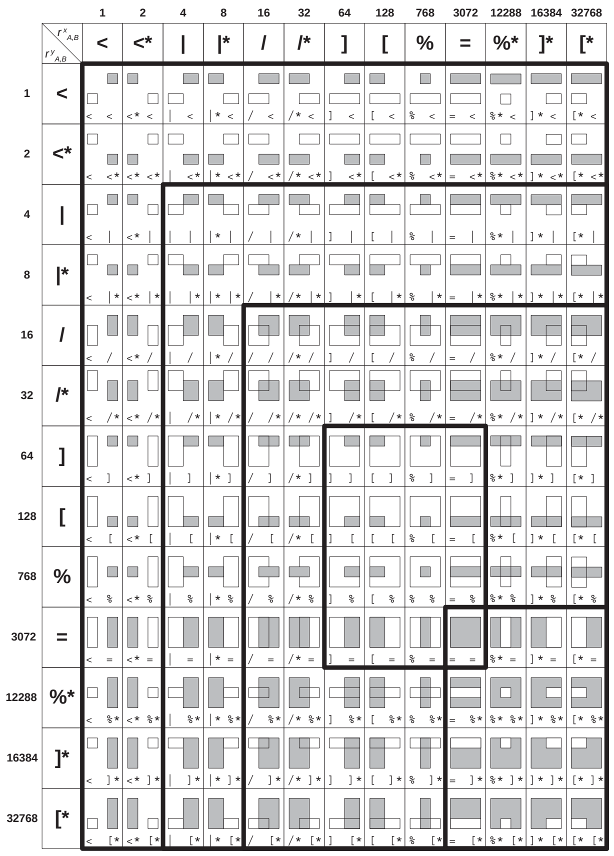


Figure 15: Three cases of bits intra-exchange: (a) Case 1; (b) Case 2; (c) Case 3.

Table 8: Category table



		1	2	4	8	16	32	64	128	768	3072	12288	16384	32768
	$r^x_{A,B}$ / $r^y_{A,B}$	<	<*	|	|*	/	/*	]	[	%	=	%*	]*	[*
1	<	< <	<* <	| <	|* <	/ <	/* <	] <	[<	% <	= <	%* <	]* <	[* <
2	<*	< <*	<* <*	| <*	|* <*	/ <*	/* <*	] <*	[<*	% <*	= <*	%* <*	]* <*	[* <*
4	|	< |	<* |	| |	|* |	/ |	/* |	] |	[|	% |	= |	%* |	]* |	[* |
8	|*	< |*	<* |*	| |*	|* |*	/ |*	/* |*	] |*	[|*	% |*	= |*	%* |*	]* |*	[* |*
16	/	< /	<* /	| /	|* /	/ /	/* /	] /	[/	% /	= /	%* /	]* /	[* /
32	/*	< /*	<* /*	| /*	|* /*	/ /*	/* /*	] /*	[/*	% /*	= /*	%* /*	]* /*	[* /*
64	]	<]	<*]	|]	|*]	/]	/*]	]]	[ ]	%]	=]	%*]	]*]	[*]
128	[	< [	<* [	| [	|* [	/ [	/* [	] [	[[	% [	= [	%* [	]* [	[* [
768	%	< %	<* %	| %	|* %	/ %	/* %	] %	[%	% %	= %	%* %	]* %	[* %
3072	=	< =	<* =	| =	|* =	/ =	/* =	] =	[=	% =	= =	%* =	]* =	[* =
12288	%*	< %*	<* %*	| %*	|* %*	/ %*	/* %*	] %*	[%*	% %*	= %*	%* %*	]* %*	[* %*
16384	]*	<]*	<*]*	|]*	|*]*	/]*	/*]*	]]*	[ ]*	%]*	=]*	%*]*	]*]*	[*]*
32768	[*	< [*	<* [*	| [*	|* [*	/ [*	/* [*	] [*	[[*	% [*	= [*	%* [*	]* [*	[* [*

3.4 Spatial Categories

According to the order of our proposed unique bit patterns, we can arrange those 169 spatial relationships into a *Category* table as shown in Table 8. Those 169 spatial relationships are grouped together into 5 different categories, highlighted by the bold lines. For example, the spatial relationships in the first, second rows and the first, second columns are the disjoin spatial category. Thus, we can use a range checking algorithm as shown in Figure 16 to distinguish the spatial category between each two objects. The parameters ubp_x and ubp_y are the unique bit patterns among x- and y-axes, respectively. The corresponding decision tree is shown in Figure 17.

```
procedure Category($ubp_x$, $ubp_y$)
1:    if ($ubp_x > 8$) and ($ubp_y > 8$) then
2:        if ($64 \leq ubp_x \leq 3072$) and ($64 \leq ubp_y \leq 3072$) then
3:            return ('contain')
4:        else if ($3072 < ubp_x \leq 32768$) and ($3072 < ubp_y \leq 32768$) then
5:                return ('belong')
6:                else return ('partial overlapping')
7:    else if ($ubp_x > 2$) and ($ubp_y > 2$)then
8:            return ('join')
9:            else return ('disjoin');
end procedure
```

Figure 16: Procedure *Category*

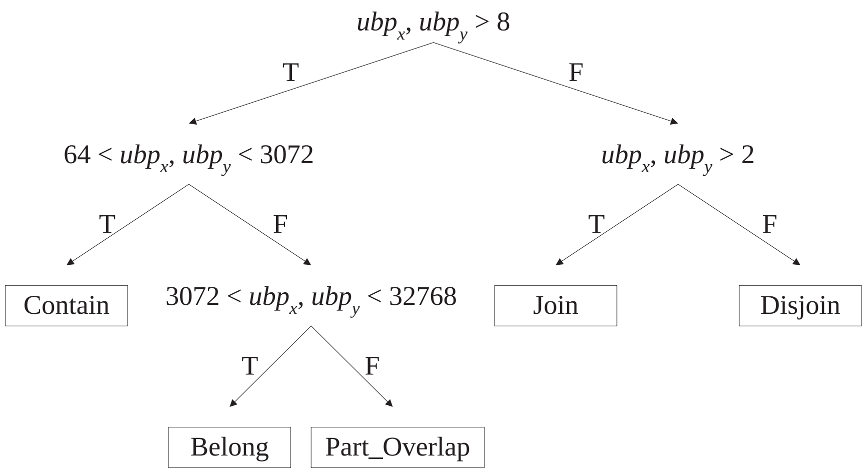


Figure 17: Decision tree of procedure *Category*

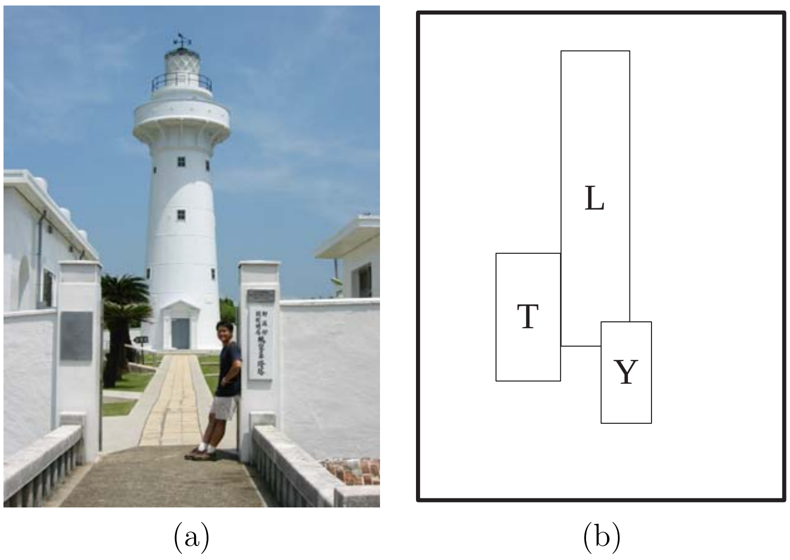


Figure 18: Example: (a) an image; (b) the symbolic representation.

3.5 The Unique Bit Pattern Matrix

Similar to previous iconic index strategies [5], we assume that there are at least two objects in an image. The spatial information between any two objects can derived. Thus, we propose a *unique-bit-pattern* matrix (UBP matrix) to preserve the spatial information of the objects in an image. Suppose an image p contains m objects and let $O = \{o_1, o_2, ..., o_m\}$. Let A be the set of 13 spatial operators $\{ <, <^*, |, |^*, [, [^*,],]^*, \%, \%^*, /, /^*, = \}$. An $m \times m$ *spatial matrix* S [8] of the image p is defined as follows:

$$S = \begin{array}{c} \\ o_1 \\ o_2 \\ \vdots \\ o_{m-1} \\ o_m \end{array} \begin{array}{c} \begin{array}{ccccc} o_1 & o_2 & \cdots & o_{m-1} & o_m \end{array} \\ \begin{bmatrix} 0 & r^y_{1,2} & \cdots & \cdots & r^y_{1,m} \\ r^x_{1,2} & 0 & \ddots & & \vdots \\ \vdots & \ddots & 0 & \ddots & \vdots \\ \vdots & & \ddots & 0 & r^y_{m-1,m} \\ r^x_{1,m} & \cdots & \cdots & r^x_{m-1,m} & 0 \end{bmatrix} \end{array}$$

where the lower triangular matrix stores the spatial information along the x-axis, and the upper triangular matrix stores the spatial information along the y-axis. That is, $r^x_{i,j}$ and $r^y_{i,j}$ are the spatial operators between objects o_i and o_j along the x- and y-axes, respectively. For the image shown in Figure 18, the corresponding spatial matrix S is shown as follows:

$$S = \begin{array}{c} \\ L \\ T \\ Y \end{array} \begin{array}{c} \begin{array}{ccc} L & T & Y \end{array} \\ \left[\begin{array}{ccc} 0 & /^* & /^* \\ |^* & 0 & /^* \\ / & < & 0 \end{array}\right] \end{array}$$

Objects L, T, and Y stand for the lighthouse, the tree, and the person, respectively. According to the assignments of the unique bit patterns for those 13 spatial operators shown in Table 7, we can transform the spatial matrix S of the image p into a *UBP matrix* M by replacing each spatial operator with its corresponding unique bit pattern (in decimal representation) as follows:

$$M = \begin{array}{c} \\ L \\ T \\ Y \end{array} \begin{array}{c} \begin{array}{ccc} L & T & Y \end{array} \\ \left[\begin{array}{ccc} 0 & 32 & 32 \\ 8 & 0 & 32 \\ 16 & 1 & 32 \end{array}\right] \end{array}$$

Then, we use the UBP matrix M to record the relative position of all objects. Moreover, the storage space of the matrix M is $3^2 \times 16$ bits. In other words, if an image p contains m objects, then we use an $m \times m$ UBP matrix with $m^2 \times 16$ bits of storage space to index the image p. Note that although the strategies recording spatial information in a matrix, *e.g.*, [3, 9], require more storage space than the strategies recording spatial information in strings, *e.g.*, [7, 18], the matrix-based strategies do similarity retrieval more efficiently than the string-based strategies [8].

3.6 Similarity Retrieval

For the similarity retrieval, because we can distinguish the spatial relationships and the spatial categories according to the UBP matrix, we can apply the same definition of the similarity measures [18], which is described in Definition 2, to determine the similarity degrees among images.

Definition 2. *Picture f' is a type-i unit picture of f, if*

1. *all objects in f' are also in f,*
2. *for any two objects A and B, the spatial bit patterns among x- and y-axes between the objects A and B in f and f' are represented as (ubp_x, ubp_y) and ($ubp_{x'}$, $ubp_{y'}$), respectively, then*

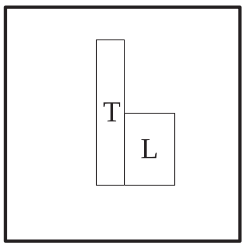


Figure 19: Symbolic representation of a query

type-0: *Category(*ubp_x*,* ubp_y*)* $=$ *Category(*$ubp_{x'}$*,* $ubp_{y'}$*);*

type-1: *(type-0) and (*$ubp_x = ubp_{x'}$ *or* $ubp_y = ubp_{y'}$*);*

type-2: $ubp_x = ubp_{x'}$ *and* $ubp_y = ubp_{y'}$*.*

For example, to find an image where there is a tree beside a lighthouse (type-0 similarity), the query may be issued by the symbolic representation as shown in Figure 19. The corresponding UBP matrix Q generated by the system is as follows:

$$Q = \begin{array}{c|cc|} & L & T \\ \hline L & 0 & 32768 \\ T & 8 & 0 \end{array}$$

Following procedure *Category* as shown in Figure 16, the spatial category between the tree and the lighthouse in the matrix Q is $Category(8, 32768)$, *i.e.*, *join.* Then, the spatial category between the two objects in the corresponding matrix M of the image shown in Figure 18 is $Category(8, 32)$, *i.e.*, *join.* Thus, the image shown in Figure 18 qualifies the query in type-0 similarity based on the Definition 2.

3.7 Deriving Indices of The Rotated and Flipped Images

We propose algorithms to derive the UBP matrices of the rotated and flipped images from the UBP matrix of the original image. Table 9 shows the definition of the symbols, M^{90}, M^{180}, M^{270}, M^H, and M^V. In our proposed algorithms, we will use those two procedures as shown in Figures 20 and 21. Procedures *Upper_Right_Triangular* and *Lower_Left_Triangular* do the *intra_exchange* operation on each entry in the upper right triangular and the lower left triangular area of a matrix, respectively. According to the

Table 9: The definition of the matrix symbols

Symbol	Definition
P	a database image
M	the UBP matrix of P
M^T	the transpose of the matrix M
M^{90}	The UBP matrix of P rotated by 90° clockwise
M^{180}	The UBP matrix of P rotated by 180° clockwise
M^{270}	The UBP matrix of P rotated by 270° clockwise
M^H	The UBP matrix of P flipped horizontally
M^V	The UBP matrix of P flipped vertically

rules of transformation shown in Table 5, we describe the algorithms to derive the matrices M^{90}, M^{180}, M^{270}, M^H, and M^V as follows.

1. Matrix M^{90}: Apply Procedure *Intra_Exchange* to each entry in the *upper right triangular* matrix of M^T, and the details are described in Procedure *M_90* shown in Figure 22.

2. Matrix M^{180}: Apply Procedure *Intra_Exchange* to each entry of M, and the details are described in Procedure *M_180* shown in Figure 23.

3. Matrix M^{270}: Apply Procedure *Intra_Exchange* to each entry in the *the lower left triangular* matrix of M^T, and the details are described in Procedure *M_270* shown in Figure 24.

4. Matrix M^H: Apply Procedure *Intra_Exchange* to each entry in the *lower left triangular* matrix of M, and the details are described in Procedure *M_H* shown in Figure 25.

5. Matrix M^V: Apply Procedure *Intra_Exchange* to each entry in the *upper right triangular* matrix of M, and the details are described in Procedure *M_V* shown in Figure 26.

We use an example shown in Figure 27 to describe the steps of deriving the matrix M^{90}. The UBP matrix M is the index of the image in the left hand side of Figure 9. Then, the rotated image by 90° clockwise is shown in the right hand side of Figure 9.

```
procedure $Upper\_Right\_Triangular(M)$ /* $M$ is an $n \times n$ UBP matrix */
1:  for $i := 1$ to $n$ do
2:      for $j := i + 1$ to $n$ do
3:          $M[i, j] := Intra\_Exchange(M[i, j])$;
4:  return $(M)$;
end procedure
```

Figure 20: Procedure *Upper_Right_Triangular*

```
procedure $Lower\_Left\_Triangular(M)$ /* $M$ is an $n \times n$ UBP matrix */
1:  for $i := 1$ to $n$ do
2:      for $j := i + 1$ to $n$ do
3:          $M[j, i] := Intra\_Exchange(M[j, i])$;
4:  return $(M)$;
end procedure
```

Figure 21: Procedure *Lower_Left_Triangular*

```
procedure $M\_90(M)$ /* $M$ is a UBP matrix */
1:  $M^T := Transpose(M)$;
2:  $M^{90} := Upper\_Right\_Triangular(M^T)$;
3:  return $(M^{90})$;
end procedure
```

Figure 22: Procedure *M_90*

```
procedure $M\_180(M)$ /* $M$ is a UBP matrix */
1:  $M^{180} := Upper\_Right\_Triangular(M)$;
2:  $M^{180} := Lower\_Left\_Triangular(M^{180})$;
3:  return $(M^{180})$;
end procedure
```

Figure 23: Procedure *M_180*

```
procedure $M\_270(M)$ /* $M$ is a UBP matrix */
1:    $M^T := Transpose(M)$;
2:    $M^{270} := Lower\_Left\_Triangular(M^T)$;
3:    return $(M^{270})$;
end procedure
```

Figure 24: Procedure *M_270*

```
procedure $M\_H(M)$ /* $M$ is a UBP matrix */
1:    $M^H := Lower\_Left\_Triangular(M)$;
2:    return $(M^H)$;
end procedure
```

Figure 25: Procedure *M_H*

```
procedure $M\_V(M)$ /* $M$ is a UBP matrix */
1:    $M^V := Upper\_Right\_Triangular(M)$;
2:    return $(M^V)$;
end procedure
```

Figure 26: Procedure *M_V*

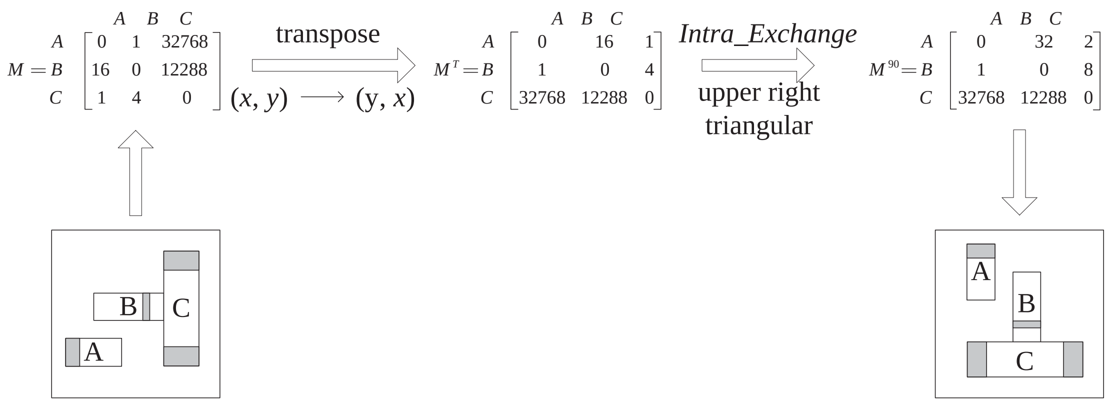

transpose
(x, y) ⟶ (y, x)
Intra_Exchange
upper right triangular
$M = \begin{array}{c|ccc} & A & B & C \\ \hline A & 0 & 1 & 32768 \\ B & 16 & 0 & 12288 \\ C & 1 & 4 & 0 \end{array}$
$M^T = \begin{array}{c|ccc} & A & B & C \\ \hline A & 0 & 16 & 1 \\ B & 1 & 0 & 4 \\ C & 32768 & 12288 & 0 \end{array}$
$M^{90} = \begin{array}{c|ccc} & A & B & C \\ \hline A & 0 & 32 & 2 \\ B & 1 & 0 & 8 \\ C & 32768 & 12288 & 0 \end{array}$
A
B
C
A
B
C

Figure 27: Process of deriving the matrix of rotated image

According to Table 5, the spatial relationships in x- and y-axes are mutually exchanged in the rotated image. Moreover, the spatial relationships in the y-axis of the rotated image are the transformed spatial relationships in the x-axis of the original image. Thus, first, we obtain the transpose of the matrix, M^T, to exchange the spatial relationships in the x- and y-axes. Then, we apply procedure *Intra_Exchange* to the upper right triangular area of the matrix M^T to change the original spatial relationships to their corresponding transformed spatial relationships. Finally, we obtain the index of the rotated image, *i.e.*, the matrix M^{90}.

4 Performance Study

In this section, first, we will analyze the time complexity of our proposed strategy. Then, we show the simulation results and discuss the properties of the results. In this simulation, we consider the CPU-time and the average search time as our performance measures. The CPU-time is the time to generate the corresponding index of the rotated or flipped image. The average search time is the time to find all qualified database for one query. Finally, we present a system prototype based on our proposed strategy.

4.1 Analysis

First, we analyze the complexity of generating the matrices, M^{90}, M^{180}, M^{270}, M^H, and M^V. Then, we analyze the complexity of searching a database image, when the query is issued in the orientation different from that of the database image.

There are two basic operations to generate the above matrices. One is procedure *Intra_Exchange*, the other is to obtain the transpose matrix, M^T. In procedure *Intra_Exchange*, from step 1 to step 5, each step is a CPU bit operation, the complexity is $O(1)$. Thus, the complexity of procedure *Intra_Exchange* is $5 \times O(1) = O(1)$. Suppose M is an $n \times n$ matrix, there are n^2 entries. To get the matrix M^T, we go through a *loop* to exchange the content of the entries, m_{ij} and m_{ji}. The complexity of the exchange is $O(1)$. The number of the exchange is $n^2/2$. Thus, the complexity of getting the matrix M^T is $n^2/2 \times O(1) = O(n^2)$. According to the algorithm to obtain the matrices M^{90}, M^{180}, M^{270}, M^H, and M^V, we list the complexity as follows.

1. M^{90}: $n^2/2 \times O(1) + O(n^2) = O(n^2)$.
2. M^{180}: $n^2 \times O(1) = O(n^2)$.
3. M^{270}: $n^2/2 \times O(1) + O(n^2) = O(n^2)$.
4. M^H: $n^2/2 \times O(1) = O(n^2)$.
5. M^V: $n^2/2 \times O(1) = O(n^2)$.

Chang *et al.* [8] have proved that to answer a query of similarity retrieval, the complexity of matching the index of the query with the index of a database image is $O(m^2)$, where m is the number of objects common to the database image and the query. Similarly, if the query is issued in the different orientation, in order not to miss the qualified database images, we need to compare another five indexes, *i.e.*, the matrices M^{90}, M^{180}, M^{270}, M^H, and M^V, with the index of an image. Thus, the complexity is $O(m^2) + 5 \times O(m^2) = O(m^2)$.

4.2 Simulation Results

We set the number of different objects appearing in the database be 100. For each object, the width and height of which are bounded between 1 and 100,000 units. We generate images with 10, 20, 30, 60, and 90 objects which are randomly chosen from those 100 different objects. In each case, 100,000 images are generated. Thus, we use 500,000 images

Table 10: The percentage of the improvement of our proposed algorithm as compared to the condition function

	10 objects	20 objects	30 objects	60 objects	90 objects
rotated by 180°	23.56%	36.45%	44.21%	47.29%	53.23%
flipped horizontally	13.64%	22.45%	28.82%	31.83%	36.72%

to calculate the average CPU-time of obtaining the index of the rotated and flipped images. Nabil *et al.* [21] and Petraglia *et al.* [22] used a condition function as shown in Figure 6 to deal with the linear transformation. This function does not show a straightforward way to change the spatial relationship to its related transformed spatial relationship. When we implement this function, we need to use conditional statements, such as *if-then-else*, to deal with the transformation. Thus, when the number of spatial relationships which need to be changed to its transformed one is huge, the process of this function is time-consuming. Instead of using the condition function, our strategy employs the *intra-exchange* bit operation to change the spatial relationship directly. In the following five cases, *Rotate in* 90°, *Rotate in* 180°, *Rotate in* 270°, *Flip horizontally*, and *Flip vertically*, we will compare the average CPU-time with different number of objects in an image. We show the two cases of the improvement of our proposed algorithm in Table 10. The more the number of objects in an image, the better improvement of our proposed algorithm is. Our proposed algorithm can reach more than 50% improvement as compared to those strategies based on the condition function, when there are 90 objects in an image.

The average search time is the time to search all the qualified images for one query. To evaluate the average search time, we prepare 100,000 images for each of the following cases in the database. We consider cases of 10, 20, 30, 60, and 90 different objects randomly chosen with the uniform distribution to appear in each image, respectively. There are 1,000 query images which contains 2 different objects. Each query is a sub-image randomly chosen from the database images. Each query image with 10%, 20%, and 50% probability to be given in the rotation ordination or the reflection direction. Then, we compare the average search time of each combination.

Table 11 shows the simulation results. For the same number of objects in a database

Table 11: The average search time (in seconds) of different combinations

	10%	20%	50%
10 objects	6.80	6.85	6.85
20 objects	17.53	18.34	18.30
30 objects	32.80	32.62	32.76
60 objects	105.63	103.28	104.80
90 objects	219.45	217.08	219.50

image, we observe that no matter what the value of the probability is, the average search time is almost the same. This is because we have to check the five cases to prevent from missing the qualified database images. We also observe that the more the number of objects in a database image, the longer the search time is. We have shown that the time complexity to do the similarity is $O(m^2)$, where m is the number of objects common to the query and the database image. Thus, the query time should be the same, when m is 2 in this simulation. However, before comparing the UBP matrix of the query with that of a database image, we need to obtain the UBP matrices of the rotated and flipped versions of the original image. Then, we have to obtain the sub-UBP matrix of the database image with the same objects information as the query. This is the reason why the more the number of objects in a database image, the longer the search time is. Note that, in fact, in the image database searching, the signature strategy [28] will be applied to prune off many unsatisfied images before comparing the indices of the candidate images with the index of the query one by one. Thus, the search time will be much less than that shown in Table 11.

4.3 The System Prototype

We use our proposed strategy, the UBP strategy, to implement a system, Interactive Image Retrieval System (IIRS), to validate the effectiveness of our proposed strategy. There are two subsystems, *image indexing* and *query*, in the IIRS. The image indexing subsystem extracts the representative objects in the images. Automatical extraction by image segmentation and object recognition is beyond our research scope [14], the subsystem provides a human-assisted interface to do it as shown in Figure 28-(a). Various techniques of image segmentation and object recognition can be found in [19]. We use this subsystem to store

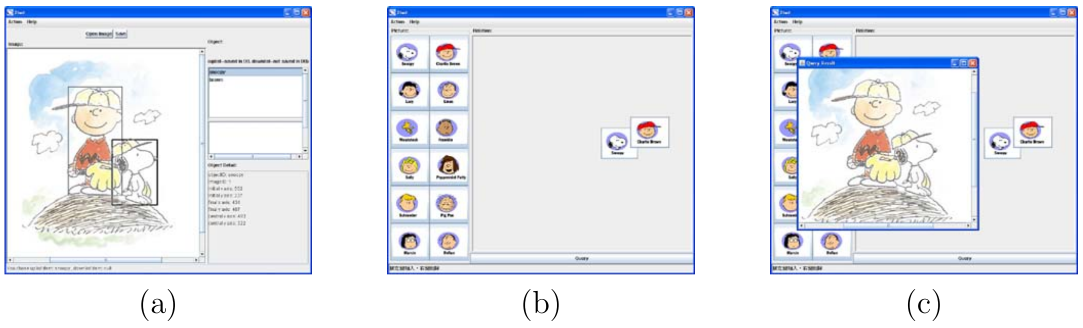


Figure 28: User interface of the prototype system: (a) the extraction of key objects; (b) a query; (c) the query result.

some images in advance, and their related UBP matrices will be generated automatically. Some images are stored in the rotated or the flipped version in order to validate our proposed strategy. The query subsystem supports queries issued in the rotation orientation or the reflection direction. For example, in Figure 28-(b), the spatial relationship between the two objects in x-axis is the flipped version as compared to the spatial relationship of the objects shown in Figure 28-(a). Then, Figure 28-(c) shows that the system can find the qualified image.

5 Conclusions

In this paper, we have presented an efficient iconic indexing strategy, *i.e.*, unique bit pattern matrix (UBP matrix), to derive the index of rotated and flipped images from the original index directly. In this way, our proposed strategy will not miss the qualified images when the query is issued in the different orientation as compared to the database images. By observing the order of the unique identifiers of the spatial relationships proposed in the UID matrix strategy, we have proposed a new order for those 13 spatial relationships to make the spatial relationship and its related transformed spatial relationship be adjacent to each other. Then, we have designed special 16-bit patterns to represent those spatial relationships. Based on these carefully designed bit patterns and our proposed bit operation, *i.e.*, *intra-exchange*, each bit pattern can be easily changed to another bit pattern of the related transformed spatial relationship. Although we reorder the spatial relationships, by viewing the bit patterns as digital numbers, we can distinguish the spatial category between each two objects by range checking algorithm. In the simulation study, we have shown that our proposed strategy makes a great improvement as compared to the strategies based on the condition function, when deriving the index of the rotated or flipped image from the original index . So far, our proposed strategy can answer queries with at least two objects with their relative position. Future work includes dealing with the query with only one object and its relative position in the whole image. For example, finding all images which have a circle at the border.

6 Acknowledgement

The authors also like to thank "Aim for Top University Plan" project of NSYSU and Ministry of Education,Taiwan, for partially supporting the research.

References

[1] J. F. Allen, "Maintaining Knowledge about Temporal Intervals," *Comm. of the ACM*, Vol. 26, No. 11, pp. 832–843, Nov. 1983.

[2] F. Cannavale, A. Casanova, M. Fraschini, and V. Savona, "Content Image Retrieval Based on Topological Information," *Journal of Visual Languages and Computing*, Vol. 15, No. 5, pp. 347–359, Oct. 2004.

[3] C. C. Chang, "Spatial Match Retrieval of Symbolic Pictures," *Journal of Information Science and Engineering*, Vol. 7, No. 3, pp. 405–422, Sept. 1991.

[4] C. C. Chang and C. F. Lee, "Relative Coordinates Oriented Symbolic String for Spatial Relationship Retrieval," *Pattern Recognition*, Vol. 28, No. 4, pp. 563–570, April 1995.

[5] S. K. Chang and E. Jungert, *Symbolic Projection for Image Information Retrieval and Spatial Reasoning.* London, U.K.: Academic Press, April 1996.

[6] S. K. Chang, E. Jungert, and G. Tortora, *Intelligent Image Database Systems.* Singapore: World Scientific Press, July 1996.

[7] S. K. Chang, Q. Y. Shi, and C. W. Yan, "Iconic Indexing by 2D Strings," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 9, No. 3, pp. 413–428, May 1987.

[8] Y. I. Chang, H. Y. Ann, and W. H. Yeh, "A Unique-ID-Based Matrix Strategy for Efficient Iconic Indexing of Symbolic Pictures," *Pattern Recognition*, Vol. 33, No. 8, pp. 1263–1276, Aug. 2000.

[9] Y. I. Chang, B. Y. Yang, and W. H. Yeh, "A Bit-Pattern-Based Matrix Strategy for Efficient Iconic Indexing of Symbolic Pictures," *Pattern Recognition Letters*, Vol. 24, No. 1–3, pp. 537–545, Jan. 2003.

[10] Y. Chen and J. Z. Wang, "A Region-Based Fuzzy Feature Matching Approach to Content-Based Image Retrieval," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 24, No. 9, pp. 1252–1267, Sept. 2002.

[11] E. A. El-kwae and M. R. Kabuka, "Efficient Content-Based Indexing of Large Image Databases," *ACM Trans. on Information Systems*, Vol. 18, No. 2, pp. 171–210, April 2000.

[12] J. B. Fraleigh and R. A. Beauregard, *Linear Algebra.* Addison Wesley, third ed., 1995.

[13] V. N. Gudivada, "θR-String: A Geometry-Based Representation for Efficient and Effective Retrieval of Images by Spatial Similarity," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 10, No. 3, pp. 504–512, May 1998.

[14] P. W. Huang and C. H. Lee, "Image Database Design Based on 9D-SPA Representation for Spatial Relations," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 16, No. 12, pp. 1486–1496, Dec. 2004.

[15] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz, "Rotaion Invariant Spherical Harmonic Representation of 3D Shape Descriptions," *Proc. of Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pp. 156–164, 2003.

[16] L. J. Latecki and R. Lakamper, “Application of Planar Shape Comparison to Object Retrieval in Image Databases,” *Pattern Recognition*, Vol. 35, No. 1, pp. 15–29, Jan. 2002.

[17] A. J. T. Lee and H. P. Chiu, “2D Z-String: A New Spatial Knowledge Representation for Image Databases,” *Pattern Recognition Letters*, Vol. 24, No. 16, pp. 3015–3026, Dec. 2003.

[18] S. Y. Lee and F. J. Hsu, “Spatial Reasoning and Similarity Retrieval of Images Using 2D C-String Knowledge Representation,” *Pattern Recognition*, Vol. 25, No. 3, pp. 305–318, March 1992.

[19] H. H. Lu, J. C. Woods, and M. Ghanbari, “Binary Partition Tree for Semantic Object Extraction and Image Segmentation,” *IEEE Trans. on Circuits and Systems for Video Technology*, Vol. 17, No. 3, pp. 378–383, March 2007.

[20] T. Marco, L. Hendrik, G. Michael, and S. Hans-Peter, “3D Acquisition of Mirroring Objects Using Striped Patterns Pages,” *Graphical Models*, Vol. 67, No. 4, pp. 233–259, 2005.

[21] M. Nabil, A. H. H. Ngu, and J. Shepherd, “Picture Similarity Retrieval Using the 2D Projection Interval Representation,” *IEEE Trans. on Knowledge and Data Engineering*, Vol. 8, No. 4, pp. 533–539, Aug. 1996.

[22] G. Petraglia, M. Sebillo, M. Tucci, and G. Tortora, “Virtual Images for Similarity Retrieval in Image Databases,” *IEEE Trans. on Knowledge and Data Engineering*, Vol. 13, No. 6, pp. 951–967, Nov./Dec. 2001.

[23] E. G. M. Petrakis, “Design and Evaluation of Spatial Similarity Approaches for Image Retrieval,” *Image and Vision Computing*, Vol. 20, No. 1, pp. 59–76, Jan. 2002.

[24] A. Rao, R. K. Srihari, L. Zhu, and A. Zhang, “A Method for Measuring the Complexity of Image Databases,” *IEEE Trans. on Knowledge and Data Engineering*, Vol. 14, No. 5, pp. 979–987, Sept. 2002.

[25] M. Saadatmand-Tarzjan and H. A. Moghaddam, “A Novel Evolutionary Approach for Optimizing Content-Based Image Indexing Algorithms,” *IEEE Trans. on Systems Man and Cybernetics Part B-Cybernetics*, Vol. 37, No. 1, pp. 139–153, Feb. 2007.

[26] E. D. Sciascio, M. Mongiello, F. M. Donini, and L. Allegretti, “Retrieval by Spatial Similarity: An Algorithm and a Comparative Evaluation,” *Pattern Recognition Letters*, Vol. 25, No. 14, pp. 1633–1645, Oct. 2004.

[27] J. C. Wong and A. Datta, “Animating Real-time Realistic Movements in Small Plants,” *Proc. of the 2nd Int. Conf. on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, pp. 182–189, 2004.

[28] W. H. Yeh and Y. I. Chang, “An Efficient Signature Extraction Method for Image Similarity Retrieval,” *Journal of Information Science and Engineering*, Vol. 22, No. 1, pp. 63–94, Jan. 2006.

[29] X. M. Zhou, C. H. Ang, and T. W. Ling, “Image Retrieval Based on Object’s Orientation Spatial Relationship,” *Pattern Recognition Letters*, Vol. 22, No. 5, pp. 469–477, April 2001.