

Algorithm for Shifting Images Stored in Peano Mask Trees

S. M. Rezaul Hoque, Shams Mahmood Imam, Mohammad Kabir Hossain, William Perrizo

Department of Computer Science and Engineering, North South University, Dhaka-1213, Bangladesh.

Department of Computer Science and Engineering, North South University, Dhaka-1213, Bangladesh.

Department of Computer Science and Engineering, North South University, Dhaka-1213, Bangladesh.

Department of Computer Science, North Dakota State University, Fargo, ND, USA.

rezahok@hotmail.com, shams_imam@hotmail.com, mkhossain@northsouth.edu, william.perrizo@mdu.nodak.edu

Abstract

Peano Trees are a data-mining ready data structure that can also be used for image compression. The ability to be able to rotate, scale or translate images is an integral part of image processing while searching through images in the image database. This paper presents an algorithm for translation of images in any direction of the two dimensional plane. The paper assumes that the images to be worked upon are stored the Peano Mask Tree format (Peano Trees are introduced below). The algorithm will provide the advantage of image processing while the image is in the Peano Tree format, it will not be necessary to generate the original bit pattern of the image to perform the task of translation.

Keywords: Data Mining, Image Translation, Peano Mask Tree, Peano Tree.

I. INTRODUCTION

There are a wide variety of applications where pattern images are required to be aligned over a reference image so that common pixels of the images are present in the same location. This process is useful in the alignment of an acquired image over a template, a time series of images of the same scene, or the separate bands of a composite image (coregistration). Two practical applications of this process are the alignment of radiology images in medical imaging and alignment of satellite images for environmental study [1].

Changing the orientation or parameters of the imaging sensor can cause differences of alignment. Hence a translation in the position of the sensor gives rise to translated images. To undo this translation effect we require translating the generated image in the opposite direction. In this paper we present such an algorithm to translate images stored in the compressed Peano Mask tree format.

II. INTRODUCTION TO PEANO TREES

A. The Peano Count Tree (P-Tree)

P-trees are a lossless, compressed, and data-mining-ready data structure. This data structure has been successfully applied in data mining applications ranging from Classification and Clustering with K-Nearest - Neighbor, to Classification with Decision Tree Induction, to Association Rule Mining [3]-[6].

The Peano Count Tree (P-tree) is a quadrant-based lossless tree representation of the original spatial data [5],[6]. The general concepts of P-trees are to recursively divide the entire spatial data into quadrants and record the count of 1-bits for each quadrant, thus forming a quadrant count tree. Using P-tree structure, all the count information can be calculated quickly. This facilitates efficient ways for data mining.

For example, a P-Tree for the given 8 x 8 image of single bits is explained below.

0	0	0	0	1	0	1	0
0	0	0	0	1	0	1	0
0	0	1	1	1	0	1	0
0	0	1	1	1	0	1	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	0
1	1	1	1	0	1	1	0

Fig. 1 8-bit by 8-bit image

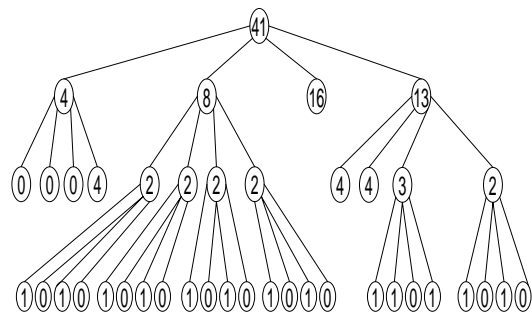


Fig. 2 P-Tree for data in Fig. 1

In this example, 41 is the number of 1's in the entire image. This root level is labeled level 0. The numbers 4, 8, 16, and 13 found at the next level (level 1) are the 1-bit counts for the four major quadrants in raster order, or Z order (upper left, upper right, lower left, and lower right). This process of storing the count of the number of ones for each of the major quadrants is repeated. When quadrants are composed entirely of 1-bits (called pure-1 quadrants), sub-trees are not needed, and these branches terminate. Similarly, quadrants composed entirely of 0-bits are called pure-0 quadrants, which also cause termination. This pattern is continued recursively using the Peano, or Z-ordering (recursive raster order-

ing), of the four sub-quadrants at each new level. Eventually, every branch terminates (since, at the “leaf” level, all quadrants are pure).

The P-tree structure can be viewed as a data-mining-ready structure as it facilitates efficient ways for data mining. P-trees have the following features:

- P-trees contain 1-count for every quadrant of every dimension.
- The P-tree for any sub-quadrant at any level is simply the sub-tree rooted at that sub-quadrant.
- A P-tree leaf sequence (depth-first) is a partial run length compressed version of the original bit-band.
- Basic P-trees can be combined to reproduce the original data (P-trees are lossless representations).
- P-trees can be partially combined to produce upper and lower bounds on all quadrant counts.
- P-trees can be used to smooth data by bottom-up quadrant purification (bottom-up replacement of mixed counts with their closest pure counts).

B. Peano Mask Tree (Pm Tree)

A variation of the P-tree data structure, the Peano Mask Tree (PM-tree), is a similar structure in which masks, rather than counts, are used. In a PM-tree, we use three-value logic to represent pure-1, pure-0, and mixed quadrants. (A 1 denotes pure-1; 0 denotes pure-0; and m denotes mixed.) The PM-tree for the previous example is also given in Figure 3. We can easily construct the original P-tree from its PM-tree by calculating the counts from the leaves to the root in a bottom-up fashion. A PM-tree is just an alternative implementation for a Peano Count tree [2].

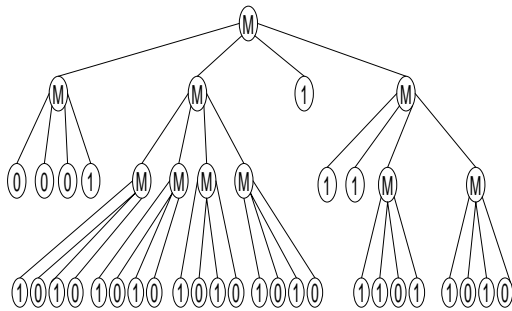


Fig. 3 PM Tree for data in Fig. 1

It is worth describing here the data structure used to implement a Peano Mask tree. The Peano Mask tree can be represented by a linked data structure in which each node is an object. Each node contains a *type* field which represents what type of node it is (0, 1 or M) and a *position* field storing which child it is of its parent (1, 2, 3 or 4). In addition each node contains fields *parent* and an array of node *child*[4] that point to the nodes corresponding to its parent and children from left to right, respectively. If a child or parent is missing, the appropriate field contains the value NIL. The root node is the

only nodes in the Peano Mask tree which has its parent field as NIL. The child nodes are numbered from 1 to 4 as their array indices, which set their position values.

III. THE TRANSLATION ALGORITHM

It can be noted that any translation may be represented as a vector comprising of two components. One component can be used to represent a right or left shift of the pixels of the image while the other component can be used to represent a up or down shift. Hence any translation consisting of integer components can be achieved via a sequence of right/left shifts followed by a sequence of up/down shifts. We present here algorithms for left and down shifts. The corresponding algorithms for right-shifts and up-shifts are symmetric and are not presented here. We assume that the image dimensions are equal powers of 2, say 2^M .

A. The Circular Left-Shift

We present here the algorithm to carry out a circular left shift operation of 2^n where $n < M$. We observe that the shift operations results in a change of the child nodes of the PM Tree at Level (M-n). Let C_{xy} represent the nodes of the PM Tree at Level (M-n-1). Hence we need to devise a method to obtain the configuration for the new children of C_{xy} . We present below a method for the new ordering of the child nodes of a node C_{xy} . As can be seen from CircularLeftShiftNode() the new arrangements of child nodes are that the children at position 2 and 4 become children at position 1 and 3, respectively. The children at position 1 and 3 of the immediate right neighbor of the current node become children at position 2 and 4, respectively. Before we can perform the shift operation on the PM Tree T we require expanding the PM Tree till Level M – n. This is achieved by recursively breaking down all Pure-1 and Pure-0 nodes of T till they are at level M – n. This is achieved via the call to function ExpandTree(). At the end of the shifting there is a corresponding call to CollapseTree(), whose function is exactly opposite that of ExpandTree().

Table I Left Shift Code

```

CircularLeftShiftTree( PMTree T, Shift-Size 2^n)
1   ExpandTree( T, M - n )
2   for each NODE n at Level (M-n-1)
3       CircularLeftShiftNode(n)
4   CollapseTree(T)

CircularLeftShiftNode( Node Cxy )
1   newCxy.child[1] ← oldCxy.child[2]
2   newCxy.child[2] ← ImmediateRightNeighbor( oldCxy ).child[1]
3   newCxy.child[3] ← oldCxy.child[4]
4   newCxy.child[4] ← IMMEDIATE-RIGHT-NEIGHBOR( oldCxy ).child[3]

ImmediateRightNeighbor( Node C )
1   Stack S

```

```

2   while (C.position≠1 OR C.position≠3)
   AND C.parent≠NIL
3     Push(S, C.position)
4     C ← C.parent
5   if C.position = 1
6     C ← C.parent.child[2]
7   else if C.position = 3
8     C ← C.parent.child[4]
9   while S ≠ ∅
10    pos ← Pop(S)
11    if pos = 4
12      newPos = 3
13    else
14      newPos = 1
15    C ← C.child[newPos]
16  return C

```

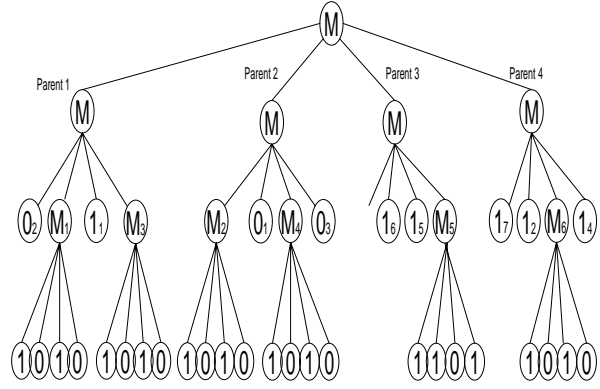


Fig. 6 Circular Shift Left Tree of 2 units

The data array when regenerated from the PM Tree of Fig. 6 is:

0	0	1	0	1	0	0	0
0	0	1	0	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	0	1	1
1	1	0	1	1	0	1	1

Fig. 7 Circular Shift Left by 2 units

Below is an example of circular left shift of size 2, i.e. 2^1 , on the data of Figure 1 and PM Tree of Fig. 3. First we present a figure of the expanded tree if we wish to perform a shift of 2 (2^1) bits.

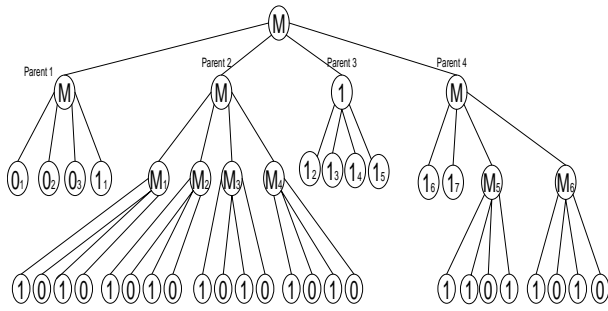


Fig. 4 Expanded Tree of Fig. 3

Next we show a figure to show the path taken by the ImmediateRightNeighbor() for a particular node.

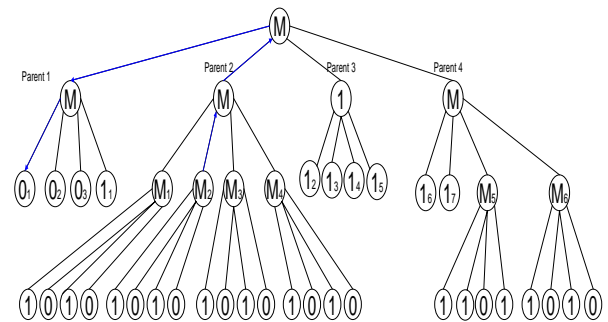


Fig. 5 Example Path followed for NextNeighborRight()

After the call to CircularLeftShiftTree(PMTree T, ShiftSize 2^1), we get the following PM Tree:

which is exactly the data array which we would have obtained if we had shifted the data bits in Figure 1 by 2 units.

B. The Circular Down-Shift

The circular down shift is similar to circular left shift but there are differences in the order of rearrangement of child nodes as well as the neighbor function that is called. As in circular left shift, we need to expand the PM Tree till Level $M - n$. The difference lies in the use of the ImmediateTopNeighbor() method.

Table II Down Shift Code

```

CircularDownShiftTree( PMTree T, ShiftSize  $2^n$  )
1   ExpandTree( T, M - n )
2   for each Node n at Level ( M-n-1 )
3     CircularDownShiftNode( n )
4   collapseTree( T )

```

```

CircularDownShiftNode( Node Cxy )
1   newCxy.child[1] ← ImmediateTopNeighbor(
   oldCxy ).child[3]
2   newCxy.child[2] ← ImmediateTopNeighbor(
   oldCxy ).child[4]
3   newCxy.child[3] ← oldCxy.child[1]
4   newCxy.child[4] ← oldCxy.child[2]

```

```

ImmediateTopNeighbor( Node C )
1   Stack S
2   while (C.position≠3 OR C.position≠4)
   AND C.parent≠NIL
3     PUSH( S , C.position )
4     C ← C.parent
5   if C.position = 3
6     C ← C.parent.child[1]
7   else if C.position = 4
8     C ← C.parent.child[2]
9   while S ≠ ∅
10    pos ← POP( S )
11    if pos = 1
12      newPos = 3
13    else
14      newPos = 4
15    C ← C.child[newPos]
16  return C

```

Below is an example of circular down shift of size 2, i.e. 2^1 , on the data of Fig. 1 and PM Tree of Fig. 3.

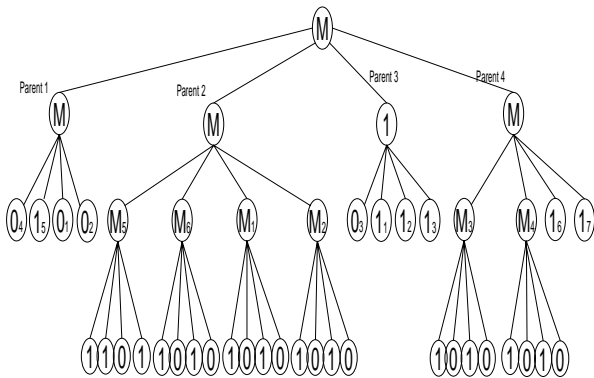


Fig. 8 Circular Shift Down Tree of 2 units

And the corresponding data bit array is:

1	1	1	1	1	1	1	0
1	1	1	1	0	1	1	0
0	0	0	0	1	0	1	0
0	0	0	0	1	0	1	0
0	0	1	1	1	0	1	0
0	0	1	1	1	0	1	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

Fig. 9 Circular Shift Down by 2 units

which again is the data array which we would have obtained if we had shifted the data bits in Fig. 1 by 2 units.

IV. ANALYSIS OF THE SHIFT ALGORITHMS

The running time of the algorithms is measured using the two parameters M , where 2^M is the one of the dimensions of the input binary data, and n , where 2^n is the size of the bits that requires to be shifted. These are the only two parameters we will use to define the time complexity of the algorithms mentioned above.

The sum of the nodes of a PM Tree up to and including the Level $M - n - 2$ is lower than the number of nodes at Level $M - n - 1$. Hence the ExpandTree() takes at most $O(4^{(M-n-2)})$ running time. Since the for loops at lines 2 of both the CircularLeftShiftTree() and CircularDownShiftTree() are entered $4^{(M-n-1)}$ times, this part of the code produces the leading term for the running time of the algorithm. In each of these loops the ImmediateNeighbor() methods are the only steps that require more than $O(1)$ time. In particular each call to ImmediateNeighbor() requires $O(M)$ time since we are moving up the tree in each of the loops, and the height of the tree is M . Hence the overall running time of any of the ShiftTree() methods is $O(M 4^{(M-n-1)}) = O(2^{\lg M + 2(M-n-1)})$. Note that this is an upper bound on the running time. A tighter bound on the running time can be achieved using a more detailed analysis.

The running time of the conventional shifting of a data array of size 2^M by 2^n units is $\Theta(2^M 2^n) = \Theta(2^{M+n})$, since we require to translate a total of 2^{M+n} bits. For the mentioned shifting algorithm to perform better than this method;

$$2^{\lg M + 2(M-n-1)} < 2^{M+n}$$

$$\Rightarrow \lg M + 2M - 2n - 2 < M + n$$

$$\Rightarrow \lg M + M < 3n + 2 \dots\dots\dots(i)$$

Whenever eq(i) is satisfied, the suggested algorithm will perform better than the conventional technique to shift any data array.

V. CONCLUSION

We are knowledgeable of the fact that mentioned algorithm can only be used with PM Trees for the shifting of bits. However, variants of Peano Trees are data mining ready structures that facilitate efficient data mining tasks. These translation algorithms can be used anywhere PM Trees are being used to recognizing patterns. Since PM Trees are both compression and lossless data structures they can be used in a variety of applications that require lossless storage of their data. Hence the popularity of proposed algorithm depends on the popularity of the use of the PM Tree data structure.

REFERENCES

[1] Morgan McGuire, "An image registration technique for recovering rotation, scale and translation pa-

- rameters,” February 19, 1998. Available at [http://www.cs.brown.edu/people/morganpapers/\(McGuire%2098\)%20Registration.pdf](http://www.cs.brown.edu/people/morganpapers/(McGuire%2098)%20Registration.pdf)
- [2] Fazle Rabbi, Mohammad Hossain, et. al. “Lossless Image Compression using P-tree,” *Proceedings of ICCIT 2003*, Dhaka, Bangladesh.
- [3] Mohammad Hossain, et. al. “Bayesian Classification for Spatial Data Using P-tree,” *Proceedings of IEEE INMIC 2004*, December 24-26, 2004 in Lahore, Pakistan.
- [4] M. K. Hossain and W. Perrizo, “Automatic fingerprint identification system using p-tree,” *Proceedings of ICCIT 2002*, Dhaka, Bangladesh.
- [5] William Perrizo, William Jockheck, Amal Perera, “Multimedia Data Mining using P-Trees,” Available at http://www-staff.it.uts.edu.au/~simeon/mdm_kdd2002/abstracts/14.html
- [6] Quang Ding, Qin Ding and William Perrizo. “Decision tree classification of spatial data streams using peano count trees,” *Proceedings of ACM Symposium on Applied Computing (SAC '02)*, Madrid, Spain, March 2002, pp. 413-417.