Proceedings of 11th International Conference on Computer and Information Technology (ICCIT 2008)
25-27 December, 2008, Khulna, Bangladesh

A Lossless Image Compression Technique Using Generic Peano Pattern Mask Tree

Mohammad Kabir Hossain[1], Shams M Imam [1], Khondker Shajadul Hasan [1], William Perrizo [2]

[1]Computer Science & Engineering, North South University, Dhaka, Bangladesh.
Computer Science, North Dakota State University, Fargo, ND, USA

Email: mkhossain@northsouth.edu, shams.mahmood@gmail.com, shajadul@northsouth.edu, william.perrizo@northsouth.edu

Abstract — **Digital Image processing has become ubiquitous in our daily life and the demands to produce and process images are ever increasing. Large amounts of space are required to store these images. Image compression techniques are in high demand as they allow reduction in this storage space. The basis for image compression, as is for most other compression techniques, is to remove redundant and unimportant data. Lossless image compression techniques retain the original information in compact form and do not introduce any errors when decompressed. In this paper, we discuss such a lossless technique using a data structure that we name "Generic Peano Pattern Mask Tree". It is an improvement over a previously discussed Lossless Image compression technique – "Peano Pattern Mask Tree". Both these structures are based on the data structure – Peano Mask Tree.**

Index terms — **Lossless Image Compression, Peano Tree, Peano Mask Tree, Peano Pattern Mask Tree, Generic Peano Pattern Mask Tree, Data mining Ready.**

I. INTRODUCTION

For many applications lossless image compression is extremely important to be able to store more images in a given amount of disk or memory space at the same time to retain the original image. Such applications include medical images, legal papers or image scans made for archival purposes, aerial photography from satellites etc. In these cases a perfect reproduction of the original image is necessary and any sort of degradation cannot be tolerated [2, 3].

There are basically two types of Image Compression methods: *Lossy* and *Lossless* [4]. Lossy compression involves the loss of some information that makes it capable of achieving much higher compression at the cost of accuracy. When the information from the compressed data using lossy technique is reconstructed, it introduces compression artifacts. In lossless compression schemes, the reconstructed image, after compression, is numerically identical to the original image. A perfect reproduction of the original image can be achieved using it.

In this paper we are proposing a method of lossless image compression technique which is an improvement over a previous work done by Shams Mahmood et. al. [1] and Fazle Rabbi et. al. [2]. In the next sections of the paper we are giving a short description of Peano Count Tree, Peano Mask Tree and Peano Pattern Mask Tree. Also we will discuss how these data structures can be used in lossless image compression. After that we will introduce the improved version of Peano Pattern Mask Tree which we call 'Generic Peano Pattern Mask Tree' and its application in lossless image compression.

II. PEANO TREES AND ITS VARIATIONS

A. The Peano Count Tree

Peano Count Tree (P-tree) is a lossless, compressed, and data-mining ready data structure. The P-tree is a quadrant-based tree representation of the original spatial data [5, 6]. The general concept of P-tree is to recursively divide the entire spatial data into quadrants and to 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. For example, a P-Tree for the given 8 × 8 image of single bit in figure 1 is explained below. Corresponding P-Tree is shown in figure 2.

```
1 1 0 0 1 0 1 1
0 1 0 0 1 0 0 1
1 1 1 1 1 0 1 0
0 1 1 1 1 0 1 0
1 1 1 1 1 1 1 1
1 1 1 1 0 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.

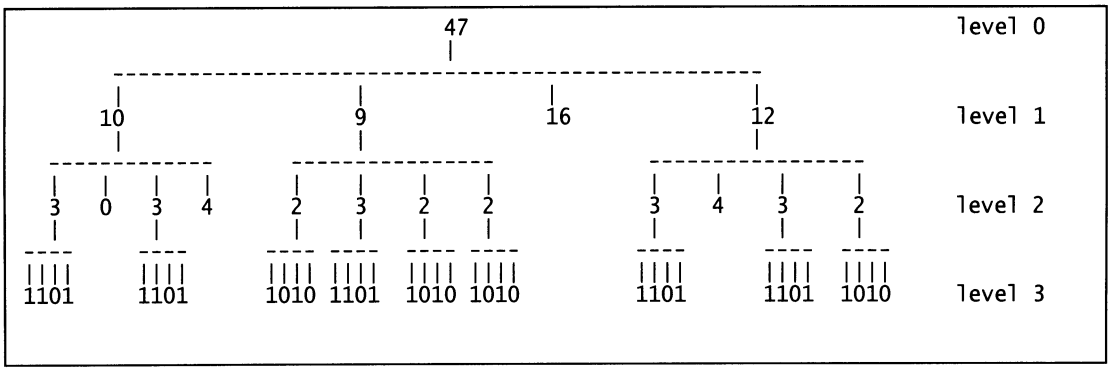

Fig. 2. P-Tree for data in figure 1.

In this example, 47 is the number of 1's in the entire image which is stored in root node which is labeled as level 0. The numbers 10, 9, 16, and 12 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 (called pure-0 quadrants), also cause termination. A quadrant which is neither pure-1 nor pure-0 is called mixed quadrant. This pattern is continued recursively using the Peano-ordering, or Z-ordering (recursive raster ordering), of the four sub-quadrants at each new level. Eventually, every branch terminates (since, at the "leaf" level, all quadrants are pure either pure-1 or pure-0).

B. The Peano Mask Tree

A variation of the P-tree data structure is the Peano Mask Tree (PM-Tree) in which masks, rather than counts, are used. In a PM-tree, we use three valued 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 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.

This PM-Tree data structure was used for compression in [2]. It is obvious that this method will work excellent when neighborhood pixels share the same bit values with high probability. This is usually the case in higher-order bits for pixels in image data. However, neighboring lower order bits have different values due to precision difference. The PM-Tree structure does not provide good compression, if any at all, for such lower order bits. This happens as quadrants that are mixed are introduced and recursive definition of data quadrants goes on till pure-1 or pure-0 quadrants are one bit long. The above mentioned paper thus chose to use only the higher four bits to construct the PM Tree. However in instances where these higher order bits lack a high degree of correlation a similar problem to the one just described might occur. A PM-Tree has three states; at least two bits are required to save the information for each node of the tree. Two bits are able to provide us with four combinations; hence, one combination remains unexploited in the PM-Tree method. This lacking was addressed in [1] which proposed a new data structure, The Peano Pattern Mask Tree.

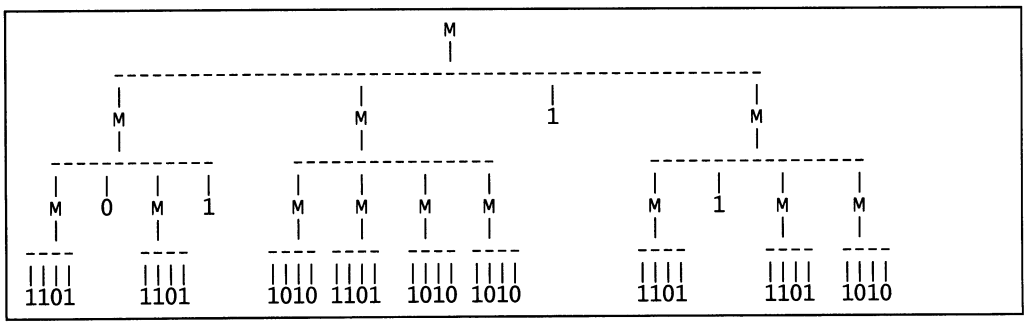

Fig. 3. PM-Tree for data in figure 1.

C. The Peano Pattern Mask Tree

The Peano Pattern Mask Tree (PPM-Tree) is another variation of the P-Tree. It shares many features of the PM-Tree and addresses some unexplored area of the PM-Tree [1]. However, instead of using three-value logic, the PPM-Tree uses four-value logic. We name this fourth logic as P, for pattern. This pattern is not fixed like pure-1 or pure-0, but rather assumes dynamic values depending on the tree constructed. In fact, the pattern state is a special case of a mixed state. An example will help us better understand the method. To construct a PPM Tree for any data set, we first need to construct a PM Tree for the same data set. Once the PM Tree is created, we search through the leaf level quadrants to look for the most frequently occurring four-bit combination. In the example for PM Tree shown in figure 3, we see that "1101" is the most frequently occurring quadrant at the leaf level. We make this our Pattern and change the original PM Tree. All M nodes having the Pattern Parent property are transformed into a P node and all children removed. This search is carried out in a bottom-up manner where nodes at the lowest levels are transformed before any node at a higher level is transformed. As a result of this, we finally get the PPM Tree as shown in figure 4.

D. Opportunities Unexplored in PPM-Tree

As mentioned earlier lower order bits for pixels in images share the same bit values with a low probability. As a result quadrants in the PM-Tree and PPM-Tree can usually be 1-bit long, which usually means no compression at all. Also due to the low probability of sharing bit values in neighborhood pixel, pure-0 and pure-1 quadrants are also rare. Hence the decision to stick with pure-0 and pure-1 as two of the four states do not help in compressing, i.e. reducing the number of levels, the tree structure. We intend to address this lacking in the PPM-Tree structure and propose a new data structure, The Generic Peano Pattern Mask Tree.

III. The Generic Peano Pattern Mask Tree

The Generic Peano Pattern Mask Tree (GPPM-Tree) is another variation of the P-Tree. It shares many features of the PPM-Tree and addresses some unexplored areas. Like the PPM Tree, the GPPM uses four-value logic. Like the PPM-Tree there is one state reserved for the mixed (M) type. However unlike the PPM-Tree there are no fixed patterns, like the pure-0 and pure-1 in the PPM-Tree. Hence there are three states available for dynamic patterns which we name pattern-1 (P1), pattern-2 (P2), and pattern-3 (P3). P1, P2 and P3 are determined dynamically during the tree is created to result in maximum compression of the tree.

The GPPM-Tree improves on the PPM-Tree by being able to choose three of the available sixteen patterns. In the PPM-Tree we were able to choose only one of available fourteen patterns, "0000" and "1111" being reserved as pure-0 and pure-1 states respectively. It can be clearly seen that the GPPM-Tree structure is a more "generic" version of the PPM-Tree as none of the available sixteen patterns are given any special privilege prior to the construction of the tree. An example will help us understand better the GPPM-Tree structure. Consider the 8-bit by 8-bit image of Figure 1. When broken down to the leaf levels in the PM-Tree structure we get the tree in figure 5. We then perform a frequency distribution of the bit patterns in the leaf nodes. (Result in Table I.)

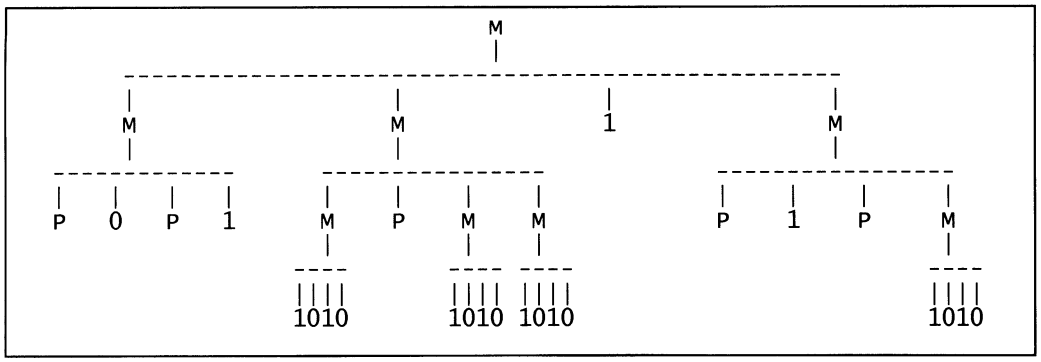

Fig. 4. PPM-Tree for data in figure 1.

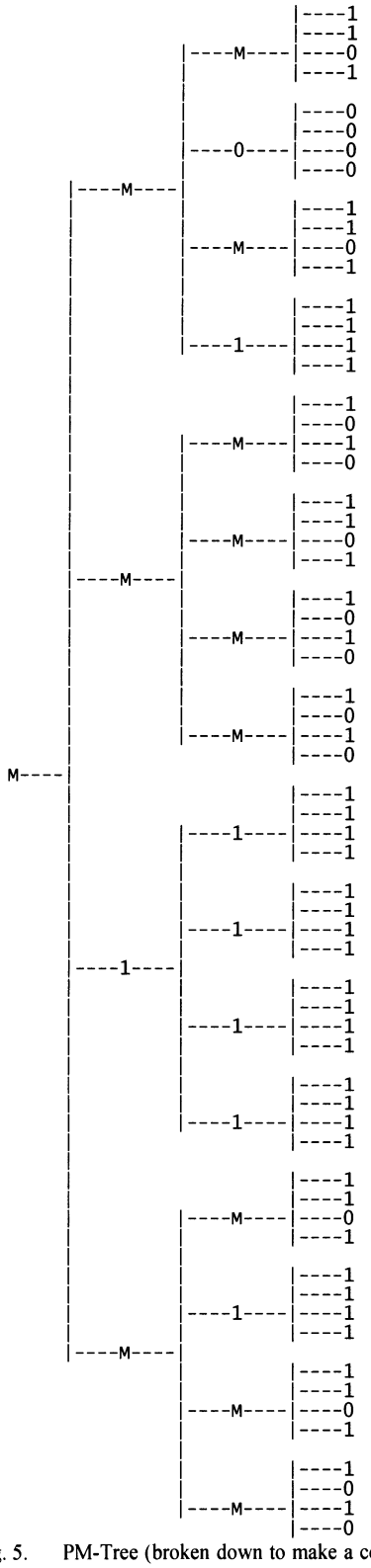

Fig. 5. PM-Tree (broken down to make a complete tree) for data in figure 1. [Vertical Layout used to fit in page]

TABLE I

FREQUENCY DISTRIBUTION OF THE BIT PATTERNS IN THE PM-TREE OF FIG. 5.

Pattern	Frequency
1111	6
1101	5
1010	4
0000	1

From Table 1 we can see that the three most frequent bit patterns are 1111, 1101 and 1010. Hence we assign these three patterns the states P1, P2 and P3 respectively to get the GPPM-Tree in figure 6

From figure 6 we can see that the GPPM-Tree has fewer nodes than the corresponding PPM-Tree, 21 nodes compared to 33 nodes. A breadth-first representation of the GPPM-Tree in figure 6 is as follows: M-M-M-P1-M-P2-M-P2-P1-P3-P2-P3-P3-P2-P1-P2-P3-0-0-0-0.

IV. PROPOSED SCHEME

Any image can be viewed as a two-dimensional array of pixels, with each pixel having various descriptive attributes. A 256 color BMP image contains descriptive attributes of size three bytes, one each for Red, Blue and Green components. Each of these color components can take an intensity value in the range from 0 to 255. Thus, BMP images require 24 bits per pixel. Representing such an image will require 24 Peano trees for each of the arrays formed from the bit positions.

The nodes of a GPPM-Tree are stored in breadth-first order using the following encoding scheme:

- For pattern-1 quadrant, store binary value 00
- For pattern-2 quadrant, store binary value 01
- For pattern-3 quadrant, store binary value 10
- For mixed quadrant, store binary value 11

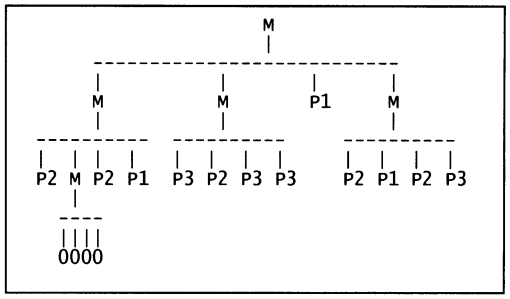

Fig. 6. GPPM-Tree for data in figure 1.

Since two bits are required to store the nodes of both PPM and GPPM Trees, for any set of data the GPPM tree will require lesser or equal space than the PPM Tree. However in the case of GPPM-Tree we also need to store the two additional patterns.

Our initial compression scheme will deal with images that have size as exact powers of 2. Consider

an n pixel image. In our proposed scheme, we first construct 24 n×n dimensional arrays using each of the 24 bits of each pixel. We then create a GPPM tree for each of these arrays. If two of the patterns for the GPPM-Tree turn out to be "0000" and "1111", then we store the tree as a PPM-Tree avoiding the overhead of storing the two additional patterns, else we store the tree as a GPPM-Tree. Next we decide if storing this GPPM-Tree/PPM-Tree for the corresponding data set from which it was generated requires lesser space than the raw data. If so then the tree is stored, else the original raw data is stored. We require a header file to keep track of which sets are stored in raw or one of the two compressed forms. Since we have three forms of storing the data we need two bits to store this information. The header file format is described below in table 2:

TABLE II

PROPOSED FILE FORMAT

Description	**Size in bytes**
Length of original image as a power of 2	1
Bits representing which segment are stored in raw or compressed form. [24 x 2 bits = 48 bits = 6 bytes]	6
Data for Red Component (8 Segments)	Variable size
Data for Green Component (8 Segments)	Variable size
Data for Blue Component (8 Segments)	Variable size

While storing the GPPM-Tree, we store the patterns in the first twelve bits (represented in bold in the figure 7) and then store the nodes of the tree in breadth-first order. Since we do this, we do not require storing a count of the number of nodes in the tree. While decoding the tree, decoding stops when all nodes are non-mixed nodes or nodes with quadrant size of 1 bit are reached. We can summarize the results of storing the raw bit pattern in Table 3.

1111 1101 1010 11 11 11 00 11 01 11 01 00 10 01 10 10 01 00 01 10 ***0000***

The first three 4-bit (bolded text) segments are the three chosen patterns. The next series of 2-bits (normal text) are representations for each of the non-1-bit length nodes/quadrants. The last series of 4-bits (emphasized text) are the patterns of the 1-bit length quadrants.

Fig. 7. Bit Representation of the GPPM-Tree of figure 6.

V. EXPERIMENTAL RESULTS

In this paper we only compared the performance of our method to the compression scheme using PPM-Tree which is mentioned in [1]. The superiority of PPM-Tree over PM-Tree is shown in [2]. A program was written for implementing the image compression scheme using GPPM-Tree. The result is summarized in Table IV. The images used in this experiment can be found at:
http://picasaweb.google.com/shams.mahmood/AdvancedPPTImages?authkey=K7eg9oaO_p4

TABLE III

SPACE REQUIRED IN STORING THE BIT PATTERN OF FIGURE 1 IN VARIOUS FORMATS

Techniques	**Number of bit requirements**
Raw data	**64** bits
GPPM-Tree	(3 x 4) + (17 x 2) + (1 x 4) bits = **50** bits.
PPM-Tree	(1 x 4) + (17 x 2) + (4 x 4) bits = **54** bits
PM-Tree	(17 x 2) + (9 x 4) bits = **70** bits.

The experimental results show that our proposed algorithm performs better than method in [1]. We see significant compression improvements have been achieved in certain cases (Jupiter_Aurora and Bangladesh). We consider this as a quite significant improvement as the PPM-Tree structure was previously thought to be a mature and optimal evolution of the P-Tree structure. We also proof with the help of our results that the GPPM-Tree is a further improvement for storing data using P-Trees

TABLE IV

EXPERIMENTAL RESULTS

Image Name	**Image Dimension**	**Size in KB**	**Size using PPM-Tree**	**Size using proposed GPPM-Tree**
Circles	128×128	48.0	28.15	27.90
Jupiter Aurora	512 x 512	768.0	381.73	376.72
Bangla desh	512 x 512	768.0	60.56	59.54
Bright	512 x 512	768.0	398.24	398.07
Dell-wallpaper	512 x 512	768.0	453.462	453.458

VI. CONCLUSION

P-tree 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 [7-10]. The P-Tree based decision tree induction method is significantly faster than existing classification methods [10, 11]. P-Tree

data structure allows computing the Bayesian probability values efficiently. Bayesian classification with P-trees has been used successfully on remotely sensed image data to predict yield in precision agriculture [7]. Experimental results showed that using p-tree techniques in an efficient association rule-mining algorithm, P-ARM has significant improvement compared to FP-growth and A-priori algorithms. [7]. Basic P-Trees can be constructed from the compressed file trivially after reconstruction of the GPPM Tree. Once the basic P-Trees have been created, we get data mining ready structure that facilitates efficient data mining tasks [1]. Thus the proposed scheme, the GPPM-Tree structure, is also a data mining ready structure providing it an upper hand over other compression techniques when used in data mining applications.

REFERENCES

[1] Shams Mahmood, Rezaul Hoque, Mohammad Kabir Hossain, William Perrizo, "A New Technique of Lossless Image Compression using PPM-Tree", in the Proceedings of 8th International Conference of Computer and Information Technology ICCIT) 2005, Dhaka, Bangladesh.

[2] Fazle Rabbi, Mohammad Kabir Hossain, William Perrizo, "Lossless Image Compression using P Tree," in the Proceedings of 6th International Conference of Computer and Information Technology ICCIT), 2003, Dhaka, Bangladesh.

[3] Erickson B J, Manduca A, Persons K R, et. al. "Evaluation of irreversible compression of digitized posterior interior chest radiographs," *J Digit Imaging 1997*; 10 3): 97 102.

[4] B. C. emuri, S. Sahni, F. Chen, C. Kapoor, C. Leonard, and J. Fitzsimmons "Lossless image compression," Available at http://citeseer.nj.nec.com/559352.html

[5] H. Samet, "The Quadtree and related hierarchical data structure," *ACM Computing Survey*, 16, 2, 1984.

[6] W. Perrizo, Peano count tree lab notes, Technical report NDSU CSOR TR 01 1, Fargo, ND, 2001.

[7] Mohammad Kabir Hossain, et. al. "Bayesian Classification for Spatial Data Using P tree," Proceedings of IEEE INMIC 2004, December 24 26, 2004 in Lahore, Pakistan.

[8] Mohammad Kabir Hossain and W. Perrizo, "Automatic fingerprint identification system using p tree," in the Proceedings of 5th International Conference of Computer and Information Technology ICCIT) 2002, Dhaka, Bangladesh.

[9] 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

[10] 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.

[11] Mohammad Kabir Hossain, et. al. "Automatic Face Recognition System using P tree and K Nearest Neighbor Classifier", in the Proceedings of 7th International Conference of Computer and Information Technology ICCIT) 2004, Dhaka, Bangladesh.