Space and Speed Advantage of pTree for Big Data Processing

Mohammad Hossain, Arjun Roy, William Perrizo

North Dakota State University

{mohammad.hossain, arjun.roy, william.perrizo}@ndsu.edu

Abstract

For so-called "big data", the time taken to process a data mining algorithm is a critical issue. Many very good algorithms are unusable in the big data environment due to the fact that the processing takes an unacceptable amount of time. Therefore, speed of processing is very important. To address the speed issue, in this paper, we use horizontal processing of vertically structured data rather than the ubiquitous vertical (scan) processing of horizontal (record) data, we use pTree, bit level, vertical data structuring. pTree technology represents and processes data differently from the ubiquitous horizontal data technologies. In pTree technology, the data is structured column-wise (into bit slices) and the columns are processed horizontally (typically across a few to a few hundred bit level columns), while in horizontal technologies, data is structured row-wise and those rows are processed vertically (often down millions, even billions of rows). P-trees are lossless, compressed and data-mining ready data structures. pTrees are lossless because the vertical bit-wise partitioning that is used in the pTree technology guarantees that all information is retained completely. There is no loss of information in converting horizontal data to this vertical format. pTrees are compressed because in this technology, segments of bit sequences which are either purely 1-bits or purely 0-bits, are represented by a single bit. This compression saves a considerable amount of space, but more importantly facilitates faster processing. pTrees are data-mining ready because the fast, horizontal data mining processes involved can be done without the need to decompress the structures first. pTree vertical data structures have been exploited in various domains and data mining algorithms, ranging from classification, clustering, association rule mining, as well as other data mining algorithms. Speed improvements are very important in data mining because many quite accurate algorithms require an unacceptable amount of processing time to complete, even with today's powerful computing systems and efficient software platforms. In this paper, we evaluate and compare the speed of various data mining algorithms when using pTree technology.

Introduction.

Predicate tree or pTree is the vertical data representation that represents the data column-by-column rather than row- by-row (which is relational data representation). It was initially developed for mining spatial data [2][4]. Since then it has been used for mining many other types of data [3][5]. The creation of pTree is typically started by converting a relational table of horizontal records to a set of vertical, compressed pTrees by decomposing each attribute in the table into separate bit vectors (e.g., one for each bit position of a numeric attribute or one bitmap for each category in a categorical attribute). Such vertical partitioning guarantees that the information is not lost.

For example, let R be a relational table consists of three numeric attributes R(A1, A2, A3). To convert it into pTree we have to convert the attribute values into binary then take vertical bit-slices of every attribute and store them in separate files. Each bit slice is considered as a pTree, which indicates the predicate if a particular bit position is zero or one. This bit slice may be compressed dividing it into binary trees recursively. Figure 1 depicts the conversion of a numerical attribute, A1, into pTrees.

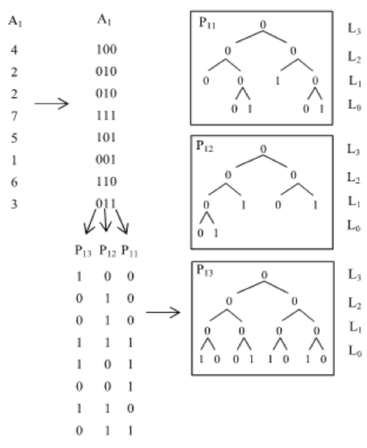

Figure 1. Construction of pTrees from attribute A1. The pTrees are built from the top down, stopping any branch as soon as purity (either purely 1-bits or purely 0- bits) is reached.

Space advantage of pTree.

Assume a dataset S consisting of N rows and n columns containing value of m bits as shown is figure 2.a

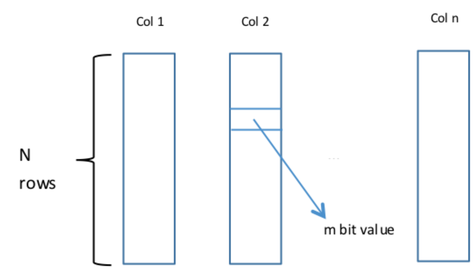

Figure 2: The data set S

So if we convert the dataset into pTrees we will get mn pTrees where the length of each tree will be N bits.

In traditional approach, let the size of the dataset be S_{trad} which we can calculate as follows:

The size of each value will be $=\left\lceil\frac{m}{8}\right\rceil$ bytes

So size of each column $= =\left\lceil\frac{m}{8}\right\rceil N$ bytes

So the size of the whole dataset, $S_{trad} = \left\lceil\frac{m}{8}\right\rceil nN$ bytes.

In pTree approach, let the size of the size of the dataset be S_{pTree}.

Size of each pTree $= \left\lceil\frac{N}{8}\right\rceil$ bytes.

So the size of the whole dataset, $S_{pTree} = mn\left\lceil\frac{N}{8}\right\rceil$ bytes.

In a large dataset where $N >> 8$, we can assume $N = 8*L$

We get $S_{trad} = 8\left\lceil\frac{m}{8}\right\rceil nL$ bytes and $S_{pTree} = mnL$ bytes

Now if $1<m<8$ then $\left\lceil\frac{m}{8}\right\rceil=1$

$$S_{pTree} = \frac{m}{8} S_{trad}$$

When m=8

$$S_{pTree} = S_{trad}$$

Again,

Now if $9<m<16$ then $\left\lceil\frac{m}{8}\right\rceil=2$

$$S_{pTree} = \frac{m}{16} S_{trad}$$

When m=16

$$S_{pTree} = S_{trad}$$

So we conclude that $S_{pTree} < S_{trad}$ and $S_{pTree} = S_{trad}$ if m is a multiple of 8.

Speed advantage of pTree

Assume that a logical operation (AND, OR, NOT, XOR) between two machine words (of size W) of memory takes T_{log} units of time. When we do such a logical operations on two pTrees of length N we actually do it on L pairs of machine words where $L = \left\lceil\frac{N}{W}\right\rceil$. So the logical operation on two pTrees takes LT_{log} unit of times.

Assume an arithmetic operation (addition, subtraction, multiplication, division) between two bytes of memory takes T_{arith} units of time.

Suppose we will do such an arithmetic operation on two columns of our previously described data set, S. For simplicity assume each value in the dataset takes 1 byte of memory.

So each column has N bytes memory and the arithmetic operation will take $T_{trad} = NT_{arith}$ units of time.

Suppose to get the same arithmetic operation using pTree we need to perform g number of logical operation on different pTrees. So the process will take $T_{pTree} = gLT_{log}$ unit of time.

For simplicity consider that N is multiple of W, so we can write

$$\frac{T_{pTree}}{T_{trad}} = \frac{g}{W}\frac{T_{log}}{T_{arith}}$$

Let's call the factor $\frac{T_{log}}{T_{arith}} = \alpha$. It is obvious that $\alpha < 1$ as because computers (GPU's) can perform the logical operations very fast. Again W represent the number of bits in a machine word which the computers (or GPU's) can handle at a time which is expected to be a large number comparing with g.

As a result the T_{pTree} will be much less than T_{trad} giving the speed gain of pTree based algorithm processing over traditional processing of the same algorithm.

Performance Analysis:

In [1] it is shown that addition of two m-bit numbers can be performed using 5m pTree operations. So if m=8 then g=40. Now using a machine word size W=64 and considering $\alpha = 1$ we can achieve 37% speed gain which is the minimum speed gain we will get.

Again if our values in the data set are in the rage of 0-15, we would need only 4 bits for each number giving us g=20. With same value of α and W we will get 68.8% speed gain. This gives us a performance lower bound that we must achieve using pTrees.

Experimental Result:

We took a data set of two columns with the values stored in 1 byte. Then we performed addition operation on the values and recorded the time as T_{trad}. Then we did the addition operation following [1] using pTrees and recorded the time T_{pTree}. Following table summarizes our findings:

Data Size in billion	T_{trad} in ms	T_{pTree} in ms	Speed gain
1	2240	200	91%
5	10720	825	92%
10	21500	1600	93%
15	32320	2600	92%
20	43400	3300	92%

Table 1: The speed gain of pTree processing on different data size.

Reference:

[1] Mohammad K Hossain, Arjun Roy, Arijit Chatterjee, William Perrizo, "Algorithms to Calculate the Manhattan (L1) Distance for Vertical Data Represented in pTrees", published in the Conference Proceedings at the ISCA 27th International Conference on Computers and Their Applications (CATA-2012), held on March 12-14, 2012, at Las Vegas, Nevada, USA.

[2] M. Khan, Q. Ding, and W. Perrizo, K-Nearest Neighbor Classification of Spatial Data Streams using P-trees, Proceedings of the PAKDD, pp. 517-528, 2002.

[3] A. Perera, A. Denton, P. Kotala, W. Jockhec, W.V. Granda, and W. Perrizo, P-tree Classification of Yeast Gene Deletion Data. SIGKDD Explorations, 4(2), pp.108-109, 2002.

[4] W. Perrizo, Peano Count Tree Technology, Technical Report NDSU-CSOR-TR-01-1, 2001.

[5] I. Rahal and W. Perrizo, An Optimized Approach for KNN Text Categorization using P-Trees. Proceedings of ACM Symposium on Applied Computing, pp. 613- 617, 2004.