

Association Rule Mining on Remotely Sensed Images Using P-trees^{1,2}

Qin Ding, Qiang Ding and William Perrizo
Computer Science Department, North Dakota State University
Fargo, ND 58105, USA
{qin.ding, qiang.ding, william.perrizo}@ndsu.nodak.edu

Abstract

Association Rule Mining, originally proposed for market basket data, has potential applications in many areas. Remote Sensed Imagery (RSI) data is one of the promising application areas. Extracting interesting patterns and rules from datasets composed of images and associated ground data, can be of importance in precision agriculture, community planning, resource discovery and other areas. However, in most cases the image data sizes are too large to be mined in a reasonable amount of time using existing algorithms. In this paper, we propose an approach to derive association rules on RSI data using Peano Count Tree (P-tree) structure. P-tree structure, proposed in our previous work [21], provides a lossless and compressed representation of image data. Based on P-trees, an efficient association rule mining algorithm P-ARM with fast support calculation and significant pruning techniques are introduced to improve the efficiency of the rule mining process. P-ARM algorithm is implemented and compared with FP-growth and Apriori algorithms. Experimental results showed that our algorithm is superior for association rule mining on RSI spatial data.

Keywords: Data Mining, Association Rule Mining, Remote Sensed Imagery (RSI)

¹ Patents are pending on the P-tree technology.

² This work is partially supported by NSF Grant OSR-9553368, DARPA Grant DAAH04-96-1-0329 and GSA Grant ACT#: K96130308.

Association Rule Mining on Remotely Sensed Images Using Peano Count Trees

1. Introduction

Association rule mining [1,2,3,4,12,19,25,26] is one of the important advances in the area of data mining. The initial application of association rule mining was on market basket data. Recently study on association rule mining has been extended to more areas, such as multimedia data [22]. An association rule is a relationship of the form $X \Rightarrow Y$, where X and Y are sets of items. X is called the antecedent and Y the consequence. An example of the rule can be, “customers who purchase an item X are very likely to purchase another item Y at the same time”. There are two primary quality measures for each rule, support and confidence. The rule $X \Rightarrow Y$ has support $s\%$ in the transaction set D if $s\%$ of transactions in D contain $X \cup Y$. The rule has confidence $c\%$ if $c\%$ of transactions in D that contain X also contain Y . The goal of association rule mining is to find all the rules with support and confidence exceeding user specified thresholds.

Remotely Sensed Imagery (RSI) data is one of the promising data areas for application of association rule mining techniques. The quantities of RSI data being collected every day from satellites, aerial sensors, telescopes and other sensor platforms are so huge that much of this data is archived before its value can be determined. Extracting the interesting rules from these datasets, in combination with other data such as ground and weather data, can provide tremendous benefits. Application areas include precision agriculture; community planning; resource discovery and management; and natural disaster prediction, detection and mitigation to mention just a few. For example, in precision agriculture, association rules can be mined from RSI data to identify crop yield potential, insect and weed infestations, nutrient requirements, and flooding damage and other phenomena. In this paper, we use as an example, the derivation of association rules from RSI data to identify high and low agricultural crop yield potential. In what is called precision agriculture, RSI data is used in mid growing season to determine where additional inputs (fertilizers, herbicides, etc.) can be effectively applied to raise the likelihood of high yield. This application serves as a good example of situations in which RSI data can be mined to alter future outcomes in a timely fashion.

An RSI image can be viewed as a 2-dimensional array of pixels. Associated with each pixel are various descriptive attributes, called “bands” in remote sensing literature [18]. For example, visible reflectance bands (Blue, Green and Red), infrared reflectance bands (e.g., NIR, MIR1, MIR2 and TIR) and possibly some bands of data gathered from ground sensors (e.g., yield quantity, yield quality, and soil attributes such as moisture and nitrate levels, etc.). All the values have been scaled to values between 0 and 255 for simplicity. The pixel coordinates in raster order constitute the key attribute. One can view such data as a relational table where each pixel is a tuple and each band is an attribute. In this paper we focus on the task of deriving association rules in which yield is specified as the rule consequent. The rule, $\text{NIR}[192,255] \wedge \text{Red}[0,63] \Rightarrow \text{Yield}[128, 255]$, which is read “Near Infrared reflectance at least 192 and Red reflectance at most 63 implies Yield will be at least 128 (e.g., bushel/acre or some normalized yield measurement)” is the type of rule expected. Such rules are useful to both producers and agribusiness communities. If low yield is predicted early in the growing year, the producer can apply additional inputs (e.g., water and nitrogen) to alter the Red and NIR and produce higher yield potential. For the agribusiness community, wide area yield estimation can improve future price prediction.

Existing algorithms do not scale well to this kind of task due to the amount of data in these images. Therefore, we propose an efficient model to perform association rule mining on RSI data. We use a bit Sequential (bSQ) format [21] to organize images and the Peano Count Tree (P-tree) structure [21] to represent bSQ files in a spatial-data-mining-ready way. P-trees are lossless representation of the image data and its histograms are in a recursive quadrant-by-quadrant arrangement. By using P-trees, association rule mining algorithm with fast support calculation and significant pruning techniques are possible.

The paper is organized as follows. Section 2 introduces the RSI data formats, including bSQ format. Section 3 summarizes the P-tree structure and its operations. Section 4 details how to derive association rules using P-trees and related pruning techniques. Experiment results and performance analysis are given in Section 5. Section 6 gives the related work and discussion. The conclusions and future work are given in Section 7.

2. RSI Data Formats

The concept of remotely sensed imagery covers a broad range of methods to include satellites, aerial photography, and ground sensors. A remotely sensed image typically contains several bands or columns of reflectance intensities. For example, Land satellite Thematic Mapper (TM) scenes contain at least seven bands (Blue, Green, Red, NIR, MIR, TIR and MIR2) while a TIFF image contains three bands (Red, Green and Blue). Each band contains a relative reflectance intensity value in the range 0-to-255 for each pixel. Ground data are collected at the surface of the earth and can be organized into images. For example, yield data can be organized into a yield map. Figure 1 gives a TIFF image and related yield map.

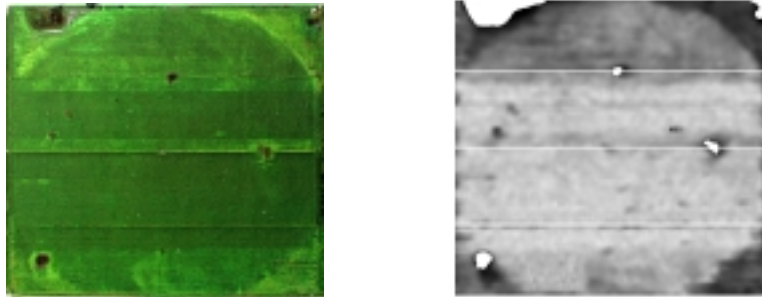


Figure 1. TIFF image (29NW082598.tiff) and related Yield Map

RSI data are collected in different ways and are organized in different formats. BSQ, BIL and BIP are three typical formats. The Band Sequential (BSQ) format is similar to the relational format. In BSQ format, each band is stored as a separate file and each individual band uses the same raster order. TM scenes are in BSQ format. The Band Interleaved by Line (BIL) format stores the data in line-major order, i.e., the first row of all bands, followed by the second row of all bands, and so on. SPOT data, which comes from French satellite sensors, are in BIL format. Band Interleaved by Pixel (BIP) is a pixel-major format. Standard TIFF images are in BIP format.

In our previous work [21], we proposed a format, called bit Sequential (bSQ), to organize spatial data. A reflectance value in a band is a number in the range 0-255 and is represented as a byte. We split each band into eight separate files, one for each bit position. There are several advantages of using bSQ formats instead of other formats. The bSQ format facilitates the representation of a precision hierarchy (from one bit precision up to eight bit precision). It also facilitates better compression. In image data, close pixels may have similar properties. By using bSQ format, close pixels may share the same bit values in high order bits. This facilitates high compression for high order bit files and brings us the idea of creating P-trees.

In Figure 2, we give a very simple illustrative example with only two data bands in a scene having only four pixels (two rows and two columns). Both decimal and binary reflectance values are given.

BAND-1				BAND-2			
254	127	37	240	(1111 1110)	(0111 1111)	(0010 0101)	(1111 0000)
14	193	200	19	(0000 1110)	(1100 0001)	(1100 1000)	(0001 0011)

BSQ format (2 files)	BIL format (1 file)	BIP format (1 file)
Band 1: 254 127 14 193 Band 2: 37 240 200 19	254 127 37 240 14 193 200 19	254 37 127 240 14 200 193 19

bSQ format (16 files, in columns)															
B11	B12	B13	B14	B15	B16	B17	B18	B21	B22	B23	B24	B25	B26	B27	B28
1	1	1	1	1	1	1	0	0	0	1	0	0	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	0	1	1	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	1	0	0	1	1

Figure 2. BSQ, BIP, BIL and bSQ formats for a two-band 2x2 image

2. Peano Count Tree (P-tree)

2.1 Basic P-trees

In this section, we summarize the P-tree structure and its operations. A P-tree is a quadrant-wise, Peano-order-run-length-compressed, representation of each bSQ file [21]. The idea is to recursively divide the entire image into quadrants and record the count of 1-bits for each quadrant, thus forming a quadrant count tree. For example, given an 8x8 bSQ file, its P-tree is as shown in Figure 3.

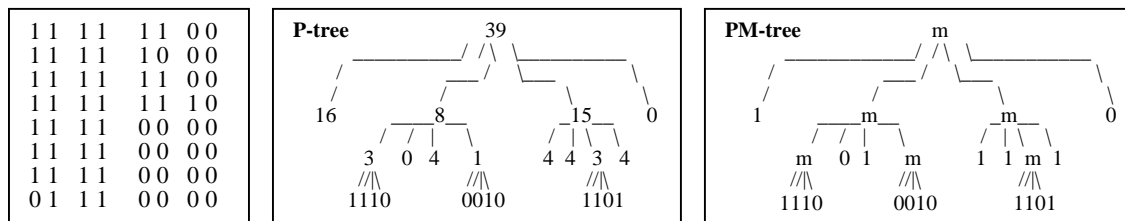


Figure 3. P-tree and PM-tree for an 8x8 bSQ file

In this example, 39 is the number of 1's in the entire image, called root count. The root level is labeled level 0. The numbers 16, 8, 15, and 0 at the next level (level 1) are the 1-bit counts for the four major quadrants in raster order (upper-left, upper-right, lower-left, lower-right). Since the first and last level-1 quadrants are composed entirely of 1-bits (called pure-1 quadrant) and 0-bits (call pure-0 quadrant) respectively, sub-trees are not needed and these branches terminate. This pattern is continued recursively using the Peano (Z-ordering) of the four sub-quadrants at each new level. Eventually, every branch terminates. If we were to expand all sub-trees, including those for pure quadrants, the leaf sequence would be the Peano-ordering of the image.

For each band (assuming 8-bit data values, though the model applies to data of any number bits), there are eight P-trees, one for each bit position.

Definition 1: A basic P-tree $P_{i,j}$ is a P-tree for the j^{th} bit of the i^{th} band.

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 a 3-value logic to represent pure-1, pure-0 and mixed (or called non-pure) quadrants (1 denotes pure-1, 0 denotes pure-0 and m denotes mixed). The PM-tree for the previous example is given in Figure 3. Since a PM-tree is just an alternative implementation for a Peano Count tree, for simplicity we will use the same term ‘‘P-tree’’ for Peano Mask Tree.

2.2 P-tree Operations

There are three basic P-tree operations: complement, AND and OR. The complement of a basic P-tree can be constructed directly from the P-tree by simply complementing the counts at each level, as shown in the example below (Figure 4). The complement of basic P-tree $P_{i,j}$ is denoted as $P_{i,j}^c$. Note that the complement of a P-tree provides the 0-bit counts for each quadrant.

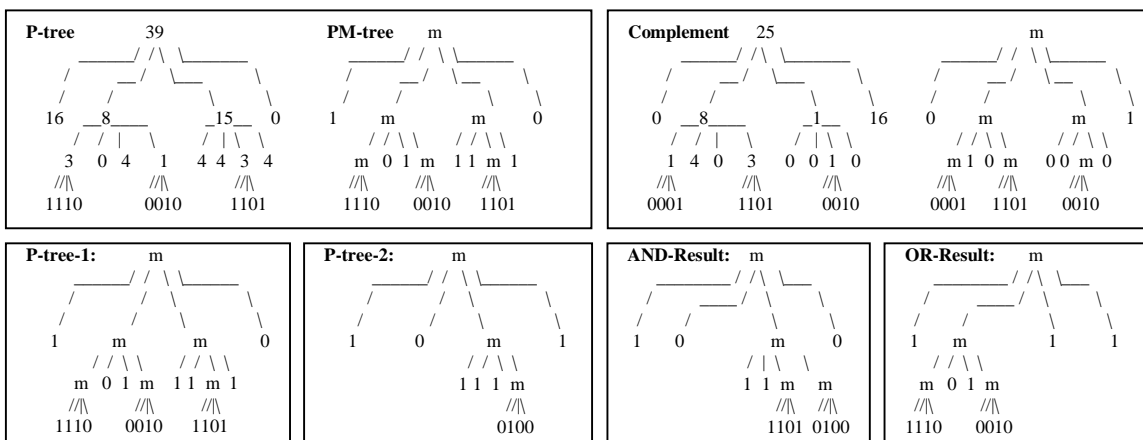


Figure 4. P-tree Operations (Complement, AND and OR)

P-tree AND/OR operations are also illustrated in Figure 4. AND is a very important and frequently used operation for P-trees. We can perform ANDing in a very straightforward way. For example, to AND a pure-1 P-tree with any P-tree X will result in X; to AND a pure-0 P-tree with any P-tree will result in a pure-0 P-tree; to AND two non-pure P-trees will result in a non-pure P-tree unless all of the four subquants result in pure-0 quadrants. OR operation can be done in a similar way.

The AND operation is just the pixel-wise AND of bits from bSQ files or their complement files. However, since such files can contain hundreds of millions of bits, shortcut methods are needed. A more efficient way for P-tree ANDing is given in [21]. The basic idea is to code P-trees in a compact, depth-first ordering of the paths to each pure-1 quadrant. Implementations of these methods have been done which allow the performance of an n-way AND of Tiff-image P-trees (1320 by 1320 pixels) in about 20 milliseconds. The process of converting data to P-trees is also time consuming unless special methods are used. For example, our methods can convert even a large TM satellite image (approximately 60 million pixels) to its basic P-trees in just a few seconds using a high performance PC computer. This is a one-time process.

2.3 Value, Tuple, Interval and Cube P-trees

By performing the AND operation on the appropriate subset of the basic P-trees and their complements, we can construct P-trees for values with more than one bit.

Definition 2: A **value P-tree** $P_i(v)$, is the P-tree of value v at band i . Value v can be expressed in 1-bit up to 8-bit precision.

Value P-trees can be constructed by ANDing basic P-trees or their complements. For example, value P-tree $P_i(110)$ gives the count of pixels with band- i bit 1 equal to 1, bit 2 equal to 1 and bit 3 equal to 0, i.e., with band- i value in the range of [192, 224). It can be constructed from the basic P-trees as:

$$P_i(110) = P_{i,1} \text{ AND } P_{i,2} \text{ AND } P_{i,3}'$$

P-trees can also represent data for any value combination from any band, even the entire tuple.

Definition 3: A **tuple P-tree** $P(v_1, v_2, \dots, v_n)$, is the P-tree of value v_i at band i , for all i from 1 to n . We have,

$$P(v_1, v_2, \dots, v_n) = P_1(v_1) \text{ AND } P_2(v_2) \text{ AND } \dots \text{ AND } P_n(v_n)$$

If value v_j is not given, it means it could be any value in Band j . For example, $P(110, ,101,001, , , ,)$ stands for a tuple P-tree of value 110 in band 1, 101 in band 3 and 001 in band 4 and any value in any other band.

Definition 4: A **interval P-tree** $P_i(v_1, v_2)$, is the P-tree for value in the interval of $[v_1, v_2]$ of band i . We have,

$$P_i(v_1, v_2) = \text{OR } P_i(v), \text{ for all } v \text{ in } [v_1, v_2].$$

Definition 5: A **cube P-tree** $P([v_{11}, v_{12}], [v_{21}, v_{22}], \dots, [v_{N1}, v_{N2}])$, is the P-tree for value in the interval of $[v_{i1}, v_{i2}]$ of band i , for all i from 1 to N .

Similar to a tuple P-tree, if the interval is a full range, i.e., from 0 to 255, this interval can be omitted.

Any value P-tree and tuple P-tree can be constructed by performing ANDing on basic P-trees and their complements. Interval and cube P-trees can be constructed by combining AND and OR operations of basic P-trees (Figure 5). All the P-tree operations, including basic operations AND, OR, COMPLEMENT and other operations such as XOR, can be performed on any kinds of P-trees defined above.

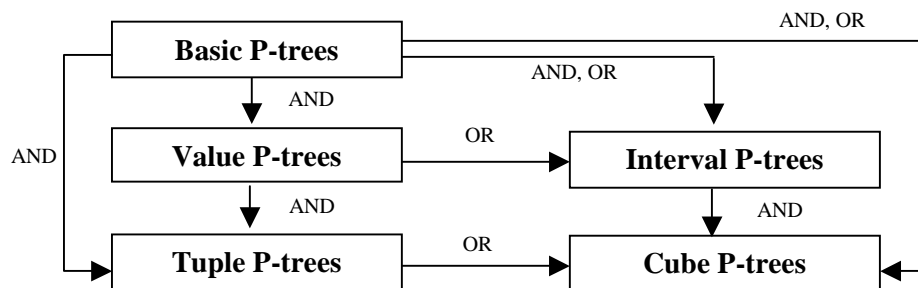


Figure 5. Basic, Value, Tuple, Interval and Cube P-trees

3. Association Rule Mining on Spatial Data Using P-trees

3.1 Data Partitioning for Numeric Data

To perform association rule mining on image data using the terminology of Association Rule Mining on market basket data ([1,2,3]), each pixel is a transaction while each band is an attribute. Every value from 0 to 255 in each band can be considered as an item (e.g., for a four-band image, the total number of items will be 1024). In this case, the mining process is expensive because of the number of items, itemsets and the database size. In addition, considering every value in each band as an item, we will get rules like “NIR=210 ^ Red=55 => Yield=188” in which values are specified single values.

To solve this problem, we can use data partitioning to partition the numeric data in images into intervals to reduce the number of items. Partitioning will also produce more interesting and general rules by using intervals instead of single values in the rules. There are several ways to partition the data, such as equi-length, equi-depth, and user-defined partitioning. Equi-length partitioning is a simple but useful method. By truncating some of the right-most bits of the values (low order or least significant bits), we can reduce the size of the itemsets dramatically without losing much information since the low order bits show only subtle differences. For example, we can truncate the right-most six bits, resulting in the set of values {00, 01, 10, 11}. Each of these values represents an equi-length partitioning of the original 8-bit value space (i.e., 00 represents the values in [0,64), 01 represents the values in [64,128), etc.).

In some cases, it would be better to allow users to partition the value space into uneven partitions. Domain knowledge can be applied in partitioning. For example, band B_i can be partitioned into $\{[0,32), [32,64) [64,96), [96,256)\}$, if it is known that there will be only a few values between 96 and 255. Applying the user's domain knowledge increases accuracy and data mining efficiency. This type of partitioning will be referred to as user-defined partitioning. Equi-depth partitioning in which each partition has approximately the same number of pixels can be done by setting the endpoints so that there are approximately the same numbers of values in each partition.

Whether partitioning is equi-length, equi-depth or user-defined, appropriate P-trees can be generated as follows. For each band, choose n partition-points, forming $n+1$ intervals. One way we can do is to replace every value in each band with its interval number in the range of $[0, n]$. Then P-trees are built on these intervalized data. For example, after partitioning data in one band into eight intervals, only three basic P-trees are needed for this band. In the case of equi-length partition, these three basic P-trees are exactly same as the original basic P-trees for the first three bits.

4.2 Deriving Association Rules Using P-trees

For RSI data, we can formulate the association rule mining model as follows. Let I be the set of all items and T be the set of all transactions. $I = \{(b,v) \mid b=\text{band}, v=\text{value (1-bit, 2-bit,..., or 8-bit)}\}$, $T = \{\text{pixels}\}$.

Definition 6: Admissible Itemsets (Asets) are itemsets of the form, $\text{Int}_1 \times \text{Int}_2 \times \dots \times \text{Int}_n = \prod_{i=1..n} \text{Int}_i$, where Int_i is an interval of values in Band_i (some of which may be unrestricted, i.e., $[0,255]$). A k -band Aset (k -Aset) is an Aset in which k of the Int_i intervals are restricted (i.e., in k of the bands the intervals are not all of $[0,255]$).

We use the notation $[a,b]_i$ for the interval $[a,b]$ in band i . For example, $[00,01]_2$ indicates the interval $[00,10]$ (which is $[0, 192]$ in decimal) in band 2.

The user may be interested in some specific kinds of rules. For an agricultural producer using precision techniques, there is little interest in rules of the type, $\text{Red} > 48 \Rightarrow \text{Green} < 134$. A physicist might be interested in such color relationships (both antecedent and consequent from color bands), but a producer is interested in rules with color antecedents and, e.g., yield consequents (i.e., observed color combinations that predict high yield or foretell low yield). Therefore, for precision agriculture applications, it makes sense to restrict our search to those rules that have a consequent in the yield band. We will refer to rule restrictions in this type as restriction to rules *of interest*, as distinct from interesting rules. Of-interest rules can be interesting or not interesting, depending on such measures as support and confidence.

Based on the Apriori algorithm [2], we first find all itemsets that are frequent and of-interest (e.g., if $B_1 = \text{Yield}$). The user may wish to restrict attention to those Asets for which Int_1 is not all of B_1 , either high-

yield or low-yield. For 1-bit data values, this means either $yield < 128$ or $Yield \geq 128$. Other threshold values can be selected using the user-defined partitioning described above.

For a frequent Aset, $B = \prod_{i=1..n} Int_i$, rules are created by partitioning $\{1..n\}$ into two disjoint sets, $\hat{A}=\{i_1..i_m\}$ and $\hat{C}=\{j_1..j_q\}$, $q+m=n$, and then forming the rule, $A \Rightarrow C$ where $A = \prod_{k \in \hat{A}} Int_k$ and $C = \prod_{k \in \hat{C}} Int_k$. As noted above, users may be interested only in rules where $q=1$ and therefore the consequents come from a specified band (e.g., $B_1=Yield$). Then there is just one rule of interest for each frequent set found and it needs only be checked for confidence.

We start by finding all frequent 1-Asets. The candidate 2-Asets are those made up of frequent 1-Asets. The candidate k-Asets are those whose (k-1)-Aset subsets are frequent. Various pruning techniques can be applied during this process based on the precision hierarchy and P-trees. Details are given later in this section. Figure 6 gives the P-ARM algorithm for mining association rules on RSI data using P-trees.

```

Procedure P-ARM
{
  Data Discretization;
  F1 = {frequent 1-Asets};
  For (k=2; Fk-1 ≠ ∅) do begin
    Ck = p-gen(Fk-1);
    For all candidate Asets c ∈ Ck do
      c.count = rootcount(c);
    Fk = {c ∈ Ck | c.count ≥ minsup}
  end
  Answer = ∪k Fk
}

```

Figure 6. P-ARM algorithm

```

insert into Ck
  select p.item1, p.item2, ..., p.itemk-1,
         q.itemk-1
  from Fk-1 p, Fk-1 q
  where p.item1 = q.item1, ..., p.itemk-2 =
        q.itemk-2, p.itemk-1 < q.itemk-1,
        p.itemk-1.group <> q.itemk-1.group

```

Figure 7. Join step in *p-gen* function

The P-ARM algorithm assumes a fixed precision, for example, 3-bit precision in all bands. In the Apriori algorithm, there is a function called “*apriori-gen*” [1] to generate candidate k-itemsets from frequent (k-1)-itemsets. The *p-gen* function in the P-ARM algorithm differs from the *apriori-gen* function in the way pruning is done. We use band-based pruning in the *p-gen* function. Since any itemsets consisting of two or more intervals from the same band will have zero support (no value can be in both intervals simultaneously), for example, support $([00,00]_1 \times [11,11]_1) = 0$, the kind of joining done in [1] is not necessary. We put different items in different groups based on bands. Each item has an associated group ID. *P-gen* only joins items from different groups. The join step in the *p-gen* function is given in Figure 7.

The *rootcount* function is used to calculate Aset counts directly by ANDing the appropriate basic P-trees instead of scanning the transaction databases. For example, in the Asets, $\{B_1[0,64), B_2[64,127)\}$, denoted as $[00,00]_1 \times [01,01]_2$, the count is the root count of $P_1(00)$ AND $P_2(01)$. This provides fast support calculation and is particularly useful for large data sets. It eventually improves the entire mining performance significantly.

4.3 Pruning Techniques

In the above algorithm, several pruning techniques have been applied or provided as options.

- **Band-based pruning**

We already mentioned that band-based pruning is used in *p-gen* function. This avoids unnecessary join operations among intervals from the same band.

- **Consequent constrain pruning**

We are only interested in of-interest rules with specified consequence, such as yield. Therefore, we only consider frequent itemsets with item in yield. This saves time by not considering the itemsets without yield even if the itemsets are frequent.

- **Bit-based pruning for multi-level rules**

There may be interest in multi-level rules [26], which means the different itemsets in the rule can have different precision in our case. A bit-based pruning technique can be applied in this case. The basic idea of bit-based pruning is that, if Aset $[1,1]_2$ (the interval $[1,1]$ in band 2) is not frequent, then the Asets $[10,10]_2$ and $[11,11]_2$ which are covered by $[1,1]_2$ cannot possibly be frequent.

Based on this we start from 1-bit precision, and try to find all the frequent itemsets. Once we find all 1-bit frequent k-Asets, we can use the fact that a 2-bit k-Aset cannot be frequent if its enclosing 1-bit k-Aset is infrequent. A 1-bit Aset encloses a 2-bit Aset if when the endpoints of the 2-bit Aset are shifted right 1-bit position, it is a subset of the 1-bit Aset. This can help to prune out the infrequent items early.

- **Root count boundary pruning**

To determine if a candidate Aset is frequent or not, we need to AND appropriate P-trees to get the root count. In fact, without performing AND operations, we can tell the boundaries for the root count by looking at the root counts of two P-trees. Suppose we have two P-trees for 8×8 bit files, with the first P-tree having root count 28 and the level-1 count 16, 12, 0 and 1, and the second P-tree having root count 36 and the level-1 count 1, 5, 16 and 14. By looking at the root level, we know the root count of ANDing result will be at most 28. If we go one more level, we can say the root count will be at most 7, calculated by $\min(16,1) + \min(12,5) + \min(0,16) + \min(1,14)$, where $\min(x,y)$ gives the minimum of x and y . If the support threshold is 30%, the corresponding A-set will not be frequent since $7/64 < 0.3$. We provide options to specify the number of levels, from 0 to 3, to check for the boundaries before performing the ANDing.

- **Adjacent interval combination using interval and cube P-trees**

P-ARM algorithm also provides an option to combine the adjacent intervals depending on user's requirements [3]. We can use interval P-trees and cube P-trees to calculate the support after combining the adjacent intervals. Fast algorithms can be used to quickly calculate the root count for interval and cube P-trees from basic P-trees. The basic idea is the root count of the AND of two P-trees with non-overlapping interval values will be the sum of root counts of these two P-trees. This avoids a lot of OR operations.

Max-support measurement proposed in [3] is used here to stop the further combination of adjacent intervals to avoid non-interesting and redundant rules.

4.4 An Example

The following simple example will be used to illustrate the method. The data contains four bands with 4-bit precision in the data values, in which B1 is Yield. We are interested to find multi-level rules so bit-based pruning technique will be applied in this example.

Field Coords (X, Y)	Class Label	Remotely Sensed Reflectance Bands		
	Yield	Blue	Green	Red
0,0	0011	0111	1000	1011
0,1	0011	0011	1000	1111
0,2	0111	0011	0100	1011
0,3	0111	0010	0101	1011
1,0	0011	0111	0100	1011
1,1	0011	0011	0100	1011
1,2	0111	0011	0100	1011
1,3	0111	0010	0101	1011
2,0	0010	1011	1000	1111
2,1	0010	1011	1000	1111
2,2	1010	1010	0100	1011
2,3	1111	1010	0100	1011
3,0	0010	1011	1000	1111
3,1	1010	1011	1000	1111
3,2	1111	1010	0100	1011
3,3	1111	1010	0100	1011

Table 2. Sample Data

The data are first converted to bSQ format. The Band 1 bit-bands as well as the basic and value P-trees are given in Figure 8. In fact, the value P-trees are not generated in advance. They are created on the fly whenever needed. Here we list some of the value P-trees to help understand the calculations.

B_{1,1} 0000 0000 0011 0111	B_{1,2} 0011 0011 0001 0011	B_{1,3} 1111 1111 1111 1111	B_{1,4} 1111 1111 0001 0011	P_{1,1} 5 0 0 1 4 0001	P_{1,2} 7 0 4 0 3 0111	P_{1,3} 16 0 0 1 1 0001 1000	P_{1,4} 11 4 4 0 3 0111
P₁(0000) 0	P₁(0100) 0	P₁(1000) 0	P₁(1100) 0	P₁(0010) 3 0 0 3 0 1110	P₁(0110) 0	P₁(1010) 2 0 0 1 1 0001 1000	P₁(1110) 0
P₁(0001) 0	P₁(0101) 0	P₁(1001) 0	P₁(1101) 0	P₁(0011) 4 4 0 0 0	P₁(0111) 4 0 4 0 0	P₁(1011) 0	P₁(1111) 3 0 0 03 0111

Figure 8. Basic and value P-trees for sample data Band 1

Assume the minimum support is 60% (requiring a count of 10) and the minimum confidence is 60%. First, we find all 1-Asets for 1-bit values from B_1 . There are two possibilities for Int_1 , $[1,1]_1$ and $[0,0]_1$, with $support([1,1]_1) = 5$ (infrequent) while $support([0,0]_1) = 11$ (frequent). Similarly, there are two possibilities for Int_2 with $support([1,1]_2) = 8$ (infrequent) and $support([0,0]_2) = 8$ (infrequent), two possibilities for Int_3 with $support([1,1]_3) = 8$ (infrequent) and $support([0,0]_3) = 8$ (infrequent), and two possibilities for Int_4 with $support([1,1]_4) = 16$ (frequent) and $support([0,0]_4) = 0$ (infrequent).

The set of 1-bit frequent 1-Asets, is $\{[0,0]_1, [1,1]_4\}$. So the set of 1-bit candidate 2-Asets, is $\{[0,0]_1 \times [1,1]_4\}$ with support 11 (root-count of $P_{1,0}$ & $P_{4,1}$) and therefore, the set of 1-bit frequent 2-Asets is $\{[0,0]_1 \times [1,1]_4\}$. The set of 1-bit candidate 3-Asets is empty.

We use Yield (B_1) as the consequent. The rule that can be formed with B_1 as the consequent is $[1,1]_4 \Rightarrow [0,0]_1$ (rule support = 11). The confidence of this rule is $11/16$ (68%). Thus, this is a strong rule.

Now consider the 2-bit case. The frequent 1-bit 1-Asets were $[0,0]_1$ and $[1,1]_4$ and the other 1-bit 1-Asets are infrequent. This means all their enclosed 2-bit subintervals are infrequent. The interval $[00,01]_1$ is identical to $[0,0]_1$ in terms of the full 8-bit values that are included, and $[00,10]_1$ is a superset of $[0,0]_1$, so both are frequent. Others in band-1 to consider are: $[00,00]_1$, $[01,01]_1$, $[01,10]_1$ and $[01,11]_1$. $[00,00]_1$ is infrequent (using $P_1(00)$, count=7). $[01,01]_1$ is infrequent (using $P_1(01)$, count=4). For $[01,10]_1$ we use $P_1(01)$ OR $P_1(10)$. The root count of $P_1(01)$ OR $P_1(10)$ is 6 and therefore $[01,10]_1$ is infrequent. The root count of $P_1(01)$ OR $P_1(10)$ OR $P_1(11)$ is 9 and therefore $[01,11]_1$ is infrequent. The only new frequent 2-bit band₁ 1-Aset is $[00,10]_1$, which does not form the support set of a rule.

Similarly, we can continue the process to the 3-bit case depending on the user's requirement.

4. Experiment Results and Performance Analysis

In this section, we compare our work with the Apriori algorithm [1], and a recently proposed efficient algorithm, FP-growth [12], which does not have the candidate generation step. The experiments are performed on a 900-MHz PC with 256 megabytes main memory, running Windows 2000. We set our algorithm to find all the frequent itemsets, not just those of interest (e.g., containing Yield) for fairness. We got identical rules by running Apriori, FP-growth and P-ARM algorithms. The images used were actual aerial TIFF images with a synchronized yield band. The data are available on the SMILEY web site [17, 24]. In our performance study, each dataset has four bands {Blue, Green, Red, Yield}. We used different image sizes up to 1320×1320 pixels.

We only store the basic P-trees for each dataset. All other P-trees (value P-trees and tuple P-trees) are created in real time as needed in the Association Rule Mining algorithm. This results in a considerable saving of space. The AND operation which produces the value and tuple P-trees as needed from basic P-tree operands is fast. Table 3 reveals the storage needs (the sizes of 8 basic P-trees for a 1320×1320 TIFF-Yield synchronized image). Note the level of compression for the high-order bits.

Bit position	Basic P-tree sizes in bytes		
	Band 1 (Red)	Band 2 (Green)	Band 3 (Blue)
1	154032	27884	91722
2	234803	128747	200845
3	254477	250175	247900
4	254751	254555	254715
5	254795	254785	254799
6	254797	254819	254823
7	254807	254809	254821
8	254807	254811	254841

Table 3. Basic P-tree Sizes

4.1 Comparison of the P-ARM with Apriori

We implemented the Apriori algorithm [2] for the TIFF-Yield datasets using equi-length partitioning. For example, using 3-bit precision, there will be 8 items (intervals) in each band for a total of 32 items. Each pixel is a transaction. For an image size 1320×1320, the total number of transactions will be 1.7M. Experiments have been done on three datasets [17, 24], having the very similar results with regard to performance.

The P-ARM algorithm is more scalable than Apriori in two ways. First, P-ARM is more scalable for lower support thresholds. The reason is that, for low support thresholds, the number of candidate itemsets will be extremely large. Thus, candidate frequent itemset generation performance degrades markedly. Figure 9 compares the results of the P-ARM algorithm and Apriori for different support thresholds.

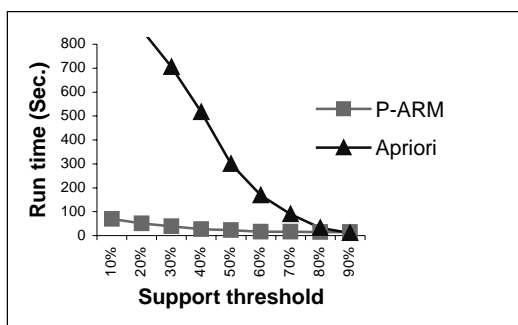


Figure 9. Scalability with support threshold

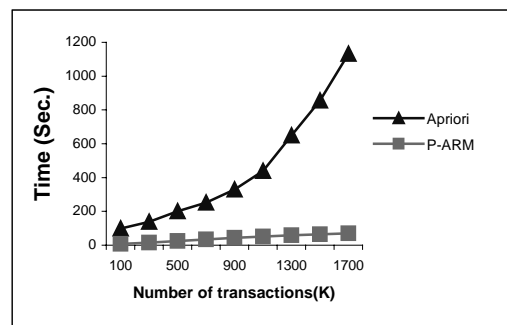


Figure 10. Scalability with number of transactions

The second conclusion is that the P-ARM algorithm is more scalable for large spatial datasets. In the Apriori algorithm we need to scan the entire database each time a support is to be calculated. This has a high cost for large databases. However, in P-ARM, we calculate the count directly from the root count of a basic P-tree and the AND program. When dataset size is doubled, only one more level is added to each basic P-tree. The additional cost is relatively small compared to the Apriori algorithm as shown in figure 10.

5.2 Comparison of the P-ARM algorithm and the FP-growth algorithm

FP-growth is an efficient algorithm for association rule mining. It uses a data structure called frequent pattern tree (FP-tree) to store compressed information about frequent patterns. We use the FP-growth object code and convert the image to the required file format in which each item is identified by an item ID. For a dataset of 100K bytes, FP-growth is fast. However, when we run the FP-growth algorithm on the TIFF image of large size, the performance falls off markedly. For large sized datasets and low support thresholds, it takes longer for FP-growth to run than P-ARM.

Figure 11 shows the experimental result of running the P-ARM and FP-growth algorithms on a 1320×1320 pixel TIFF-Yield dataset (in which the total number of transactions is about 1,700,000). In these experiments, we have used 2-bits precision and equi-length partitioning.

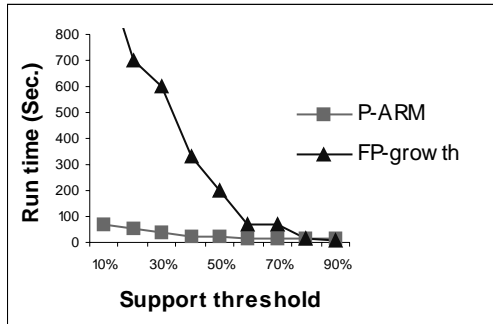


Figure 11. Scalability with support threshold

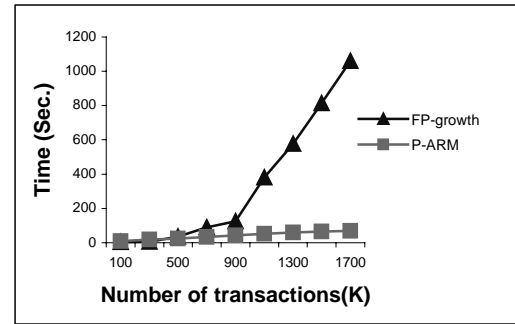


Figure 12. Scalability with number of transactions

From the above figures we can see that both the P-ARM and the FP-growth algorithms run faster than the Apriori algorithm for most cases. For large image datasets and low support threshold, the P-ARM algorithm runs faster than the FP-tree algorithm.

Our test suggests that the FP-growth algorithm runs quite fast for datasets with transactions numbering less than 500,000 ($|D| < 500K$). For the larger datasets, the P-ARM algorithm gives a better performance. This result is presented in Figure 12 (where the support threshold was set at 10%).

Figure 13 shows how the number of precision bits used affects the performance of the P-ARM algorithm. The more precision bits are used, the greater the number of items, the longer the mining time. By adjusting the number of precision bits, we can balance the trade-off between the number of rules discovered and the quality of the rules.

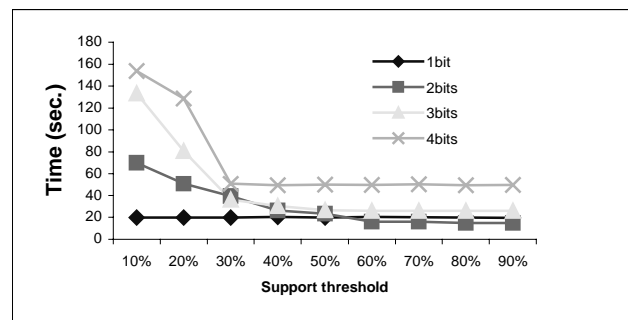


Figure 13. Performance of the P-ARM algorithm with respect to the number of precision bits

5.3 Sample Rules

We run P-ARM algorithm on the dataset of 29NW083097.tiff and associated yield, nitrate and moisture [24]. Some of the sample rules are listed below. Assume minsup = 30%, minconf = 60%, we use 2-bit equi-length partition:

Red[0,63] ^ Green[0,63] => Yield[128,191]	(sup=34.99%, conf=99.38%)
Red[0,63] ^ Green[0,63] ^ Blue[0-63] => Yield[128,191]	(sup=34.76%, conf=75.98%)
Red[0,63] ^ Green[0,63] ^ Moisture[128,192] => Yield[128,191]	(sup=31.23%, conf=68.65%)
Nitrate[192, 255] ^ Moisture[128,192] => Yield[192,255]	(sup=30.02%, conf=78.27%)

Such rules are very useful in precision agriculture. If low yield is predicted early in the growing year, the producer can apply additional inputs, such as water and nitrogen, to alter the Red, Green, Blue, Nitrate and Moisture to produce higher yield potential.

5. Related work and Discussions

Remotely Sensed Imagery data belongs to the category of spatial data. Some works on spatial data mining are [13,14,15], including Association Rule Mining on Spatial data [16]. A spatial association rule

is a rule indicating that a certain association relationship exists among a set of spatial and possibly some nonspatial predicates [16]. An example rule might be “most big cities in Canada are close to the Canada-U.S. border”. In these works, spatial data mining is performed with the perspective of spatial locality. Our work is different from this point of view. The patterns among spectral bands and yield do not necessarily exist on close pixels only. They can exist in any part of the image. The rules generated in this way are useful to the agricultural producer.

Some work has been done on association rule mining on image data. One topic is association rule mining based on image content [22, 23]. In these studies, the problem is to find associations rules about the size, color, texture, shape of the images and object identification in images. Our work is different in the sense that we perform pixel-by-pixel mining on images.

In our previous work [21], we proposed an approach to derive high confidence rules using P-trees. A data cubed is generated based on P-trees to find the rules with high confidence. In this paper, we target on general association rule mining problems by considering both the support and confidence of the rules.

The P-tree structure is related to some other structures including quadtrees [6,7,9] (and its variants point quadtrees [9] and region quadtrees [6]), and HHcode [10]. The similarities between P-trees, quadtrees and HH-Codes are that they are quadrant based. The difference is that P-trees include occurrence counts. P-trees are not indexes, but are representations of the dataset itself. P-trees incorporate inherent data compression. Moreover, P-trees contain useful information to facilitate efficient association rule mining. Wavelets also provide a good way for compressing data, but unlike P-trees, they do not facilitate pixel-by-pixel mining.

The work in [20] presents the ideas of using AD-trees (All-Dimension trees) for machine learning with large datasets. AD-trees are a sparse data structure for representing the cached counting statistics for categorical datasets. The AD-tree records information for all dimensions. P-trees record count information for separate bits and bands. This information is small, simple and complete.

The P-trees are based on Peano ordering. Peano ordering was selected for several reasons. Compared to raster ordering, Peano ordering has better spatial clustering properties. Peano ordering facilitates compression of the count tree structures due to the probability that neighboring pixels will be similar. Hilbert ordering shares this property, however, experimental results showed that Hilbert ordering made the count tree structure more complex while having almost the same compression ratio as Peano ordering.

6. Conclusion and Future Work

In this paper, we propose a new model to derive association rules from Remotely Sensed Imagery. In our model, bSQ format and Peano Count Tree (P-tree) structure are used to organize and represent RSI data. The P-tree structure is a space efficient, lossless, data mining ready structure for spatial datasets. P-trees facilitate advantage, such as fast support calculation and new pruning techniques with early algorithm exit, for association rule mining. Similarly, P-trees can also facilitate the fast calculation of other measurements, such as “interest” and “conviction” defined in [19].

P-trees have potential applications in the areas other than precision agriculture. These areas include flood prediction and monitoring, community and regional planning, virtual archeology, mineral exploration, Bioinformatics, VLSI design and environmental analysis and control. Our future work includes extending P-trees to these and other application areas. Another interesting direction is the application of P-trees on sequential pattern mining.

Acknowledgement

We would like to express our thanks to Dr. Jiawei Han of Simon Fraser University for providing us the FP-growth object code.

References

- [1] R. Agrawal, T. Imielinski, and A. Swami, “Mining Association Rules Between Sets of Items in Large Database”, SIGMOD 93.
- [2] R. Agrawal and R. Srikant, “Fast Algorithms for Mining Association Rules,” VLDB 94.
- [3] R. Srikant and R. Agrawal, "Mining Quantitative Association Rules in Large Relational Tables", SIGMOD 96.

- [4] Jong Soo Park, Ming-Syan Chen and Philip S. Yu, "An effective Hash-Based Algorithm for Mining Association Rules," SIGMOD 95.
- [5] V. Gaede and O. Gunther, "Multidimensional Access Methods", Computing Surveys, 30(2), 1998.
- [6] H. Samet, "The quadtree and related hierarchical data structure". ACM Computing Survey, 16, 2, 1984.
- [7] H. Samet, "Applications of Spatial Data Structures", Addison-Wesley, Reading, Mass., 1990.
- [8] H. Samet, "The Design and Analysis of Spatial Data Structures", Addison-Wesley, 1990.
- [9] R. A. Finkel and J. L. Bentley, "Quad trees: A data structure for retrieval of composite keys", Acta Informatica, 4, 1, 1974.
- [10] HH-codes. Available at <http://www.statkart.no/nlhdb/iveher/hhtext.htm>
- [11] S. W. Golomb, "Run-length encoding", IEEE Trans. On Information Theory, 12(3), July 1966.
- [12] J. Han, J. Pei and Y. Yin, "Mining Frequent Patterns without Candidate Generation", SIGMOD 2000.
- [13] M. Ester, H. P. Kriegel, J. Sander, "Spatial Data Mining: A Database Approach", SSD 1997.
- [14] K. Koperski, J. Adhikary, J. Han, "Spatial Data Mining: Progress and Challenges", DMKD 1996.
- [15] M. Ester, A. Frommelt, H. P. Kriegel, J. Sander, "Spatial Data Mining: Database Primitives, Algorithms and Efficient DBMS Support", Data Mining and Knowledge Discovery, 4(2/3).
- [16] K. Koperski, J. Han, "Discovery of Spatial Association Rules in Geographic Information Databases", SSD 1995.
- [17] SMILEY project. Available at <http://midas.cs.ndsu.nodak.edu/~smiley>
- [18] Remote Sensing Tutorial. Available at <http://rst.gsfc.nasa.gov/Front/tofc.html>
- [19] S. Brin et al, "Dynamic Itemset Counting and Implication Rules for Market Basket Data", SIGMOD 97.
- [20] A. Moore, M. Soon Lee, "Cached Sufficient Statistics for Efficient Machine Learning with Large Datasets", Journal of Artificial Intelligence Research, 8 (1998).
- [21] William Perrizo, Qin Ding, Qiang Ding and Amalendu Roy, "Deriving High Confidence Rules from Spatial Data using Peano Count Trees", Springer-Verlag, LNCS 2118, July 2001.
- [22] O. R. Zaiane, J. Han and H. Zhu, "Mining Recurrent Items in Multimedia with Progressive Resolution Refinement", ICDE'2000.
- [23] C. Ordonez and E. Omiecinski, "Discovering Association Rules based on Image Content", Proceedings of the IEEE Advances in Digital Libraries Conference (ADL'99).
- [24] TIFF image data sets. Available at <http://midas-10.cs.ndsu.nodak.edu/data/images/>
- [25] R. Miller and Y. Yang, "Association Rule Mining over Interval Data", SIGMOD 97.
- [26] J. Han and Y. Fu, "Discovery of Multiple-level Association Rules from Large Databases", VLDB 95.