Fast Distance Metric Based Data Mining

Techniques Using P-trees:

k-Nearest-Neighbor Classification and k-Clustering

A Thesis
Submitted to the Graduate Faculty
Of the North Dakota State University
Of Agriculture and Applied Science

By

Md Abdul Maleq Khan

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

December 2001

Fargo, North Dakota

TABLE OF CONTENTS

ABSTRACT …… iv

ACKNOWLEDGEMENT …… v

DEDICATION ……. vi

LIST OF FIGURES ……..... vii

CHAPTER 1: GENERAL INTRODUCTION ……. 1

CHAPTER 2: DISTANCE METRICS AND THEIR BEHAVIOR ……. 4

2.1 Definition of a Distance Metric ……... 4

2.2 Various Distance Metrics …… 5

2.3 Neighborhood of a Point Using Different Distance Metrics …… 14

2.4 Decision Boundaries for the Distance Metrics …….. 16

CHAPTER 3: P-TREES AND ITS ALGEBRA & PROPERTIES …… 18

3.1 P-trees and Its Algebra …….. 18

3.2 Properties of P-trees …….. 21

3.3 Header of a P-tree File …….. 25

3.4 Dealing with Padded Zeros ……... 26

CHAPTER 4: PAPER 1

K-NEAREST NEIGHBOR CLASSIFICATION ON SPATIAL DATA STREAMS USING P-TREES …… 28

Abstract ……... 28

4.1 Introduction ……... 29

4.2 Classification Algorithm ……... 32

4.2.1 Expansion of Neighborhood …….. 35

4.2.2 Computing the Nearest Neighbors ……. 38

4.2.3 Finding the Plurality Class Among the Nearest Neighbors ……... 40

4.3 Performance Analysis ……..……. 41

4.4 Conclusions ……... 46

References ……... 43

CHAPTER 5: PAPER 2

FAST K-CLUSTERING ALGORITHM ON SPATIAL DATA USING P-TREES ……. 48

Abstract ……... 48

5.1 Introduction ……... 49

5.2 Review of the Clustering Algorithms ……... 52

5.2.1 k-Means Algorithm …… 52

5.2.2 The Mean-Split Algorithm ……. 52

5.2.3 Variance Based Algorithm ……. 54

5.3 Our Algorithm …….. 54

5.3.1 Computation of Sum and Mean from the P-trees …….. 57

5.3.2 Computation of Variance from the P-trees …… 61

5.4 Conclusion …… 64

References ……... 64

CHAPTER 6: GENERAL CONCLUSION ……..……... 66

BIBLIOGRAPHY ……... 67

ABSTRACT

Khan, Md Abdul Maleq, M.S., Department of Computer Science, College of Science and Mathematics, North Dakota State University, December 20001. Fast Distance Metric Based Data Mining Techniques Using P-trees: k-Nearest-Neighbor Classification and k-Clustering. Major Professor: Dr. William Perrizo.

Data mining on spatial data has become important due to the fact that there are huge volumes of spatial data now available holding a wealth of valuable information. Distance metrics are used to find similar data objects that lead to develop robust algorithms for the data mining functionalities such as classification and clustering. In this paper we explored various distance metrics and their behavior and developed a new distance metric called HOB distance that provides an efficient way of computation using P-trees. We devised two new fast algorithms, one k-Nearest Neighbor Classification and one k-Clustering, based on the distance metrics using our new, rich, data-mining-ready structure, the Peano-count-tree or P-tree. In these two algorithms we have shown how to use P-trees to perform distance metric based computation for data mining. Experimental results show that our P-tree based techniques outperform the existing techniques.

ACKNOWLEDGEMENT

I would like to thank my adviser, Dr. William Perrizo, for his guidance and encouragement in developing the ideas during this research work. I would also like to thank the other members of the supervisory committee, Dr. D. Bruce Erickson, Dr. John Martin and Dr. Marepalli B. Rao for taking time from their busy schedule to serve on the committee. Thanks to Qin Ding for her help in running the experiments. Finally special thanks to William Jockheck for his help in writing and in using the correct and appropriate structures of language.

DEDICATION

Dedicated to the memory of my father, Nayeb Uddin Khan.

LIST OF FIGURES

Figure Page

2.1: *two-dimensional space showing various distance between points X and Y* …..…… 6

2.2: *neighborhood using different distance metrics for 2-dimensional data points* …...… 14

2.3: *Decision boundary between points A and B using an arbitrary distance metric d* 14

2.4: *Decision boundary for Manhattan, Euclidian and Max distance* ……... 15

2.5: *Decision boundary for HOB distance* ……..... 15

3.1 *Peano ordering or Z-ordering* ……..… 17

3.2: *8-by-8 image and its P-tree (P-tree and PM-tree)* ……. 17

3.3: *P-tree Algebra* ……. 19

3.4: *Header of a P-tree file* ……. 23

4.1: *Closed-KNN set* …….…….. 30

4.2: *Algorithm to find closed-KNN set based on* HOB *metric* ……..…….. 36

4.3(a): *Algorithm to find closed-KNN set based on Max metric (Perfect Centering)* ...…. 37

4.3(b): *Algorithm to compute value P-trees* ……... 37

4.4: *Algorithm to find the plurality class* ……...…. 38

4.5: *Accuracy of different implementations for the* 1997 *and* 1998 *datasets* …… 40

4.6: *Comparison of neighborhood for different distance metrics* ……...……..….. 41

4.7: *Classification time per sample for the different implementations for the* 1997 *and* 1998 *datasets. Both of the size and classification time are plotted in logarithmic scale* …… 42

CHAPTER 1: GENERAL INTRODUCTION

Data mining is the process of extracting knowledge from a large amount of data. Data mining functionalities include data characterization and discrimination, Association rule mining, classification and prediction, cluster analysis, outlier analysis, evolution analysis etc. We focus on classification and cluster analysis. Classification is the process of predicting the class of a data object whose class label is unknown using a model derived from a set of data called a training dataset. The class labels of all data objects in the training dataset are known. Clustering is the process of grouping objects such that the objects in the same group are similar and two objects in different groups are dissimilar. Clustering can also be viewed as the process of finding equivalence classes of the data objects where each cluster is an equivalence class.

Distance metrics play an important role in data mining. Distance metric gives a numerical value that measures the similarity between two data objects. In classification, the class of a new data object having unknown class label is predicted as the class of its similar objects. In clustering, the similar objects are grouped together. The most common distance metrics are Euclidian distance, Manhattan distance, Max distance. There are also some other distances such as Canberra distance, Cord distance and Chi-squared distance that are also used for some specific purposes.

In chapter 2, we discussed various distance metrics and their behavior. The neighborhood and decision boundaries for different distance metrics are depicted graphically. We developed a new distance metric called **Higher Order Bit (HOB) distance**. Chapter 2 includes a proof that HOB satisfies the property of a distance metric.

A P-tree is a quadrant based data structure that stores the count information of 1 bits of the quadrants and its sub-quadrants successively level by level. We construct one P-tree for each bit position. For example, from the first bits of the first attribute of all data points, we construct the P-tree $P_{1,1}$. The count information stored in P-trees makes it data-mining-ready and thus facilitates the construction of fast algorithms for data mining. P-trees also provide a significant compression of data. This can be an advantage when fitting data into main memory.

In chapter 3, we review the P-tree data structure and its various forms including the logical AND/OR/COMPLEMENT operations on P-trees. We reveal some useful and interesting properties of P-trees. A header for P-tree files to form a generalized P-tree structure is included.

In chapter 4, we include a paper: "K-Nearest Neighbor (KNN) Classification on Spatial Data Streams Using P-Trees". Instead of using a traditional KNN set we build a closed-KNN. The definition of our new closed KNN is given in section 4.2. We develop two efficient algorithms using P-trees based on HOB and Max distance. The experimental results using different distance metrics have been included.

In chapter 5, we included another paper: "Fast k-Clustering of Spatial Data Using P-trees". We develop a new efficient algorithm for k-clustering. In k-clustering, we need to compute the mean and variance of the data samples. Theorems including their proofs to compute mean and variance from P-trees without scanning databases have been given in section 5.3. *k*-clustering using P-trees involves computation of interval P-trees. An optimal algorithm to compute interval P-trees has also been included. These algorithms, theorems

and our fast P-tree AND/OR operations construct a very fast clustering method that does not require any database scan.

CHAPTER 2: DISTANCE METRICS AND THEIR BEHAVIOR

2.1 Definition of a Distance Metric

A distance metric measures the dissimilarity between two data points in terms of some numerical value. It also measures similarity; we can say that more distance less similar and less distance more similar.

To define a distance metric, we need to designate a set of points, and give a rule, $d(X, Y)$, for measuring distance between any two points, X and Y, of the space. Mathematically, a distance metric is a function, d, which maps any two points, X and Y in the n-dimensional space, into a real number, such that it satisfies the following three criteria.

Criteria of a Distance Metric

a) *d(X, Y) is* ***positive definite****:* If the points X and Y are different, the distance between them must be positive. If the points are the same, then the distance must be zero. That is, for any two points X and Y,

   i. if $(X \neq Y)$, $d(X, Y) > 0$

   ii. if $(X = Y)$, $d(X, Y) = 0$

b) *d(X, Y) is* ***symmetric***: The distance from X to Y is the same as the distance from Y to X. That is, for any two points X and Y,

$$d(X, Y) = d(Y, X)$$

c) *d(X, Y) satisfies* ***triangle inequality***: The distance between two points can never be more than the sum of their distances from some third point. That is, for any three points *X*, *Y* and *Z*,

$$d(X, Y) + d(Y, Z) \geq d(X, Z)$$

2.2 Various Distance Metrics

The presence of the pixel grid makes several so-called distance metrics possible that often give different answers for the distance between the same pair of points. Among the possibilities, Manhattan, Euclidian, and Max distance metrics are common.

Minkowski Distance

The general form of these distances is the weighted Minkowski distance. Considering a point, X, in n-dimensional space as a vector $<x_1, x_2, x_3, \ldots, x_n>$,

the weighted Minkowski distance, $d_p(X,Y) = \left\{ \sum_{i=1}^{n} w_i |x_i - y_i|^p \right\}^{\frac{1}{p}}$

Where, p is a positive integer,

x_i and y_i are the i^{th} components of *X* and *Y*, respectively.

w_i (≥ 0) is the weight associated to the i^{th} dimension or i^{th} feature.

Associating weights allows some of the features dominate the others in similarity matching. This is useful when it is known that some features of the data are more important than the others. Otherwise, the Minkowski distance is used with $w_i = 1$ for all *i*. This is also known as the $\boldsymbol{L_p}$ **distance.**

$$d_p(X,Y) = \left\{ \sum_{i=1}^{n} |x_i - y_i|^p \right\}^{\frac{1}{p}}$$

Manhattan Distance

When $p = 1$, the Minkowski distance or the $\boldsymbol{L_1}$ **distance** is called the Manhattan distance.

The Manhattan distance, $d_1(X,Y) = \sum_{i=1}^{n} |x_i - y_i|$

It is also known as the **City Block distance**. This metric assumes that in going from one pixel to the other it is only possible to travel directly along pixel grid lines. Diagonal moves are not allowed.

Euclidian Distance

With $p = 2$, the Minkowski distance or the $\boldsymbol{L_2}$ **distance** is known as the Euclidian distance.

The Euclidian distance, $d_2(X,Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$

This is the most familiar distance that we use, in our daily life, to find the shortest distance between two points (x_1, y_1) and (x_2, y_2) in a two dimensional space; that is

$$d_2(X,Y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

Max Distance

When $p = \infty$, the summation, in the Minkowski distance or $\boldsymbol{L_\infty}$ **distance**, is dominated by the largest difference, $|x_k - y_k|$ for some k ($1 \le k \le n$) and the other differences are negligible. Hence L_∞ distance becomes equal to the maximum of the differences.

The Max distance, $d_\infty(X,Y) = \max_{i=1}^{n} |x_i - y_i|$

Max distance is also known as the **chessboard distance**. This metric assumes that you can make moves on the pixel grid as if you were a 'King' making moves in chess, i.e. a diagonal move counts the same as a horizontal move.

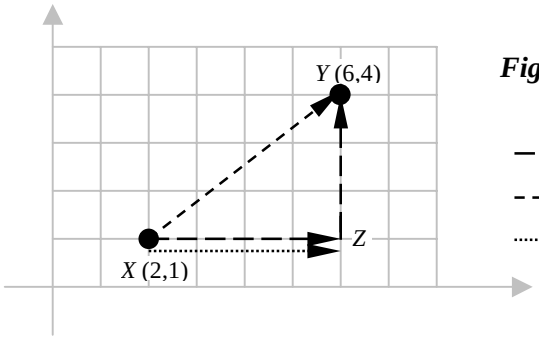


***Figure* 2.1:** *two-dimensional space showing various distance between points X and Y.*

— — — *Manhattan,* $d_1(X,Y) = XZ + ZY = 4+3 = 7$

- - - - - *Euclidian,* $d_2(X,Y) = XY = 5$

····· *Max,* $d_\infty(X,Y) = \text{Max}(XZ, ZY) = XZ = 4$

It is clearly understood from the figure 2.1 that $d_1 \geq d_2 \geq d_\infty$ for any two points *X* and *Y*.

Theorem 1: *For any two points X and Y, the Minkowski distance metric (or L_p distance),* $d_p(X,Y) = \left\{ \sum_{i=1}^{n} |x_i - y_i|^p \right\}^{\frac{1}{p}}$ *is a monotone decreasing function of p; that is* $d_p \geq d_q$ if $p < q$.

Proof: $d_p(X,Y) = \left\{ \sum_{i=1}^{n} |x_i - y_i|^p \right\}^{\frac{1}{p}} = \left\{ \sum_{i=1}^{n} z_i^p \right\}^{\frac{1}{p}}$, *letting* $|x_i - y_i| = z_i$, where $z_i \geq 0$

Assuming $X \neq Y$ and $\max_{i=1}^{n}\{z_i\} = z_k$, we see $z_k \neq 0$.

Let $\frac{z_i}{z_k} = \alpha_i$, then $0 \leq \alpha_i \leq 1$ and

$$d_p(X,Y) = \left\{ z_k^p \sum_{i=1}^{n} \alpha_i^p \right\}^{\frac{1}{p}} = z_k \left\{ \sum_{i=1}^{n} \alpha_i^p \right\}^{\frac{1}{p}} \text{ and } \sum_{i=0}^{n} \alpha_i^p \geq 1, \text{ since } \sum_{i=0}^{n} z_i^p \geq z_k^p$$

Similarly, $d_q(X,Y) = z_k \left\{ \sum_{i=1}^{n} \alpha_i^q \right\}^{\frac{1}{q}}$ and $\sum_{i=0}^{n} \alpha_i^q \geq 1$

Now $0 \le \alpha_i \le 1 \Rightarrow \alpha_i^p \ge a_i^q \Rightarrow \sum_{i=1}^{n} \alpha_i^p \ge \sum_{i=0}^{n} a_i^q$, since $p < q$

$\Rightarrow \left\{ \sum_{i=1}^{n} \alpha_i^p \right\}^q \ge \left\{ \sum_{i=0}^{n} a_i^q \right\}^p$, since $\sum_{i=0}^{n} \alpha_i^p \ge 1$ and $\sum_{i=0}^{n} \alpha_i^{p+1} \ge 1$ and $p < q$

$\Rightarrow z_k \left\{ \sum_{i=1}^{n} \alpha_i^p \right\}^{\frac{1}{p}} \ge z_k \left\{ \sum_{i=0}^{n} a_i^q \right\}^{\frac{1}{q}}$

That is $d_p(X,Y) \ge d_q(X,Y)$

Again when $X = Y$, $d_p(X,Y) = d_q(X,Y) = 0$

Therefore, for any X and Y, $d_p(X,Y) \ge d_q(X,Y)$.

Corollary 1(a): $L_0 : d_0(X,Y) = \lim_{p \to 0} z_k \left\{ \sum_{i=1}^{n} \alpha_i^p \right\}^{\frac{1}{p}} = z_k \lim_{p \to 0} \left\{ \sum_{i=1}^{n} \alpha_i^p \right\}^{\frac{1}{p}}$, which is not defined.

Corollary 1(b): $L_\infty : d_\infty(X,Y) = \lim_{p \to \infty} z_k \left\{ \sum_{i=1}^{n} \alpha_i^p \right\}^{\frac{1}{p}} = z_k \lim_{p \to \infty} \left\{ \sum_{i=1}^{n} \alpha_i^p \right\}^{\frac{1}{p}} = z_k = \max_{i=1}^{n} |x_i - y_i|$.

Canberra Distance

Canberra distance is defined by $d_c(X,Y) = \sum_{i=1}^{n} \frac{|x_i - y_i|}{x_i + y_i}$

Squared Cord Distance

Squared cord distance is defined by $d_{sc}(X,Y) = \sum_{i=1}^{n} \left(\sqrt{x_i} - \sqrt{y_i} \right)^2$

Squared Chi-squared distance

Squared Chi-squared distance is defined by $d_{chi}(X,Y) = \sum_{i=1}^{n} \frac{(x_i - y_i)^2}{x_i + y_i}$

Higher Order Bit (HOB) Distance

In this paper we propose a new distance metric called **HOB distance** that provides an efficient way of computation using P-trees. HOB distance is defined for the data where each component of a data point is an integer such as reflectance values of a pixel. We use similarity in the most significant bit positions between band values of two pixels. We consider only the most significant consecutive bit positions starting from the left most bit, which is the highest order bit. Consider the following two 8-bit values, x_1 and y_1, represented in binary. The 1st bit is the most significant bit and 8th bit is the least significant bit.

Bit position:	1	2	3	4	5	6	7	8
x_1:	0	1	1	0	1	0	0	1
y_1:	0	1	1	1	1	1	0	1

	1	2	3	4	5	6	7	8
x_1:	0	1	1	0	1	0	0	1
y_2:	0	1	1	0	0	1	0	0

These two values are similar in the three most significant bit positions, 1st, 2nd and 3rd bits (011). After they differ (in 4th bit), we don't consider anymore lower order bit positions though x_1 and y_1 have identical bits in the 5th, 7th and 8th positions. Since we are looking for closeness in values, after differing in some higher order bit positions, similarity in some lower order bit is meaningless with respect to our purpose. Similarly, x_1 and y_2 are identical in the four most significant bits (0110). Therefore, according to our definition, x_1 is more similar to y_2 than to y_1.

Definition 1: *The* HOB *similarity between two integers A and B is defined by*

$$\text{HOB}(A, B) = \max_{s=0}^{m}\{s : \forall i(1 \le i \le s \Rightarrow a_i = b_i)\}$$

where a_i and b_i are the i^{th} bits of A and B respectively and m ($m \geq 1$) is the number of bits in binary representations of the values. All values must be represented using the same number of bits. Or in other words,

HOB(A, B) = s, *where for all* $i \leq s$, $a_i = b_i$ *and* ($a_{s+1} \neq b_{s+1}$ or $s = m$),

Definition 2: *The* HOB *distance between two integers A and B is defined by*

$d_v(A, B) = m - \text{HOB}(A, B)$.

Definition 3: *The* **HOB *distance*** *between two points X and Y is defined by*

$$d_{HOBS}(X,Y) \text{ or } d_H(X,Y) = \max_{i=1}^{n}\{d_v(x_i,y_i)\} = \max_{i=1}^{n}\{m - \text{HOB}(x_i,y_i)\}$$

Where n is the number of dimensions. x_i and y_i are the i^{th} components of X and Y respectively.

Definition 4: When A and B are equal, A and B are identical in all m bits. Again when A and B are identical in all m bit, A and B are equal. We say,

$A = B$ if and only if $\forall i(1 \leq i \leq m \Rightarrow a_i = b_i)$

Lemma 1: *For any two integers A and B,* HOB(A, B) *is defined and* $0 \leq \text{HOB}(A, B) \leq m$.

Proof: If $s = 0$, for any i, $1 \leq i \leq s \Leftrightarrow$ FALSE, hence $\forall i(1 \leq i \leq s \Rightarrow a_i = b_i) \Leftrightarrow$ TRUE

Therefore, the set $\{s : \forall i(1 \leq i \leq s \Rightarrow a_i = b_i)\}$ contains the element 0 irrespective of A and B.

Hence, for any two integers A and B, $\max_{s=0}^{m}\{s : \forall i(1 \leq i \leq s \Rightarrow a_i = b_i)\}$ has a value, i.e. HOB(A, B) is defined.

$\max_{s=0}^{m}\{s\} = m$ and $\min_{s=0}^{m}\{s\} = 0$, then $0 \leq \max_{s=0}^{m}\{s : \forall i(1 \leq i \leq s \Rightarrow a_i = b_i)\} \leq m$

That is $0 \le \text{HOB}(A, B) \le m$.

Lemma 2: $0 \le d_v(A, B) \le m$.

Proof: $0 \le \text{HOB}(A, B) \le m$ (Lemma 1)

$\Rightarrow m - 0 \ge m - \text{HOB}(A, B) \ge m - m$

$\Rightarrow m \ge d_v(A, B) \ge 0$

$\Rightarrow 0 \le d_v(A, B) \le m$

Lemma 3: $\text{HOB}(A, A) = m$ *and* $d_v(A, A) = 0$

Proof: $\text{HOB}(A, A) = \max_{s=0}^{m}\{s : \forall i(1 \le i \le s \Rightarrow a_i = a_i)\} = \max_{s=0}^{m}\{s\} = m$

$d_v(A, A) = m - \text{HOB}(A, A) = 0$

Lemma 4: *If* $A \ne B$, $\text{HOB}(A, B) < m$ *and* $d_v(A, B) > 0$

Proof: (Proof by contradiction)

Assume, $\text{HOB}(A, B) = m$

$\Rightarrow \forall i(1 \le i \le m \Rightarrow a_i = b_i)$ $\langle \text{Definition 1} \rangle$

$\Rightarrow A = B$ $\langle \text{Definition 4} \rangle$

But it is given that $A \ne B$ (contradiction), Therefore $\text{HOB}(A, B) \ne m$.

Again, $\text{HOB}(A, B) \le m$ $\langle \text{Lemma 1} \rangle$

Hence, $\text{HOB}(A, B) < m$

And $m - \text{HOB}(A, B) > 0$

$\Rightarrow d_v(A, B) > 0$

Theorem 3: HOB *distance is positive definite i.e. for any two points X and Y,*

a) $if\,(X = Y),\ d_H(X,Y) = 0$

b) $if\,(X \neq Y),\ d_H(X,Y) > 0$

Proof: a) $d_H(X, Y)$

$= d_H(X, X) \quad \langle X = Y \rangle$

$= \max_{i=1}^{n}\{m - \text{HOBS}(x_i, x_i)\} = \max_{i=1}^{n}\{m - m\} \quad \langle \text{Lemma 3} \rangle$

$= \max_{i=1}^{n}\{0\} = 0$

b) $X \neq Y \Rightarrow$ there exits some k such that $x_k \neq y_k$

$\Rightarrow d_v(x_k, y_k) > 0 \quad \langle \text{Lemma 4} \rangle$

$\Rightarrow \max_{i=1}^{n}\{d_v(x_i, y_i)\} > 0 \quad \langle \text{For any } i,\ d_v(x_i, y_i) \geq 0,\ \text{Lemma 2} \rangle$

That is $d_H(X,Y) > 0$.

Lemma 5: $\text{HOB}(A, B) = \text{HOB}(B, A)$.

Proof: $\text{HOB}(A, B)$

$= \max_{s=0}^{m}\{s : \forall i(1 \leq i \leq s \Rightarrow a_i = b_i)\}$

$= \max_{s=0}^{m}\{s : \forall i(1 \leq i \leq s \Rightarrow b_i = a_i)\}$

$= \text{HOB}(B, A)$

Theorem 4: HOB *distance is symmetric i.e. for any two points X and Y,* $d_H(X,Y) = d_H(Y, X)$.

Proof: $d_H(X,Y)$

$= \max_{i=1}^{n}\{m - \text{HOB}(x_i, y_i)\}$

$= \max_{i=1}^{n}\{m - \text{HOB}(y_i, x_i)\}$ $\langle \text{Lemma 5} \rangle$

$= d_H(Y, X)$

Lemma 6: *For any three integers A, B and C,* $d_v(A,B) + d_v(B,C) \geq d_v(A,C)$.

Proof: Let $\text{HOB}(A,B) = p$ and $\text{HOB}(B,C) = q$.

That is, $\max_{s=0}^{m}\{s : \forall i(1 \leq i \leq s \Rightarrow a_i = b_i)\} = p$ and $\max_{s=0}^{m}\{s : \forall i(1 \leq i \leq s \Rightarrow b_i = c_i)\} = q$

$\Rightarrow \forall i(1 \leq i \leq p \Rightarrow a_i = b_i) \wedge \forall i(1 \leq i \leq q \Rightarrow b_i = c_i)$

$\Rightarrow \forall i(1 \leq i \leq \min(p,q) \Rightarrow (a_i = b_i) \wedge (b_i = c_i))$

$\Rightarrow \forall i(1 \leq i \leq \min(p,q) \Rightarrow a_i = c_i)$

$\Rightarrow \text{HOB}(A,C) = \max_{s=0}^{m}\{s : \forall i(1 \leq i \leq s \Rightarrow a_i = c_i)\} \geq \min(p,q)$

$\Rightarrow m - \text{HOB}(A,C) \leq m - \min(p,q)$

That is $m - \min(p,q) \geq d_v(A,C)$.

We know that $(m-p) + (m-q) \geq \max(m-p, m-q) = m - \min(p,q) \geq d_v(A,C)$.

That is $d_v(A,B) + d_v(B,C) \geq d_v(A,C)$ $\langle \text{Definition 2} \rangle$

Theorem 5: HOB *distance satisfies triangle inequality, i.e. for any three points X, Y and Z,*

$d_H(X,Y) + d_H(Y,Z) \geq d_H(X,Z)$.

Proof: Assume $d_H(X,Z) = \max_{i=1}^{n}\{d_v(x_i, z_i)\} = d_v(x_k, z_k)$, for some integer *k*, where $1 \leq k \leq n$.

Now $\max_{i=1}^{n}\{d_v(x_i, y_i)\} \geq d_v(x_k, y_k)$ and $\max_{i=1}^{n}\{d_v(y_i, z_i)\} \geq d_v(y_k, z_k)$

$$\Rightarrow \max_{i=1}^{n}\{d_v(x_i,y_i)\} + \max_{i=1}^{n}\{d_v(y_i,z_i)\} \geq d_v(x_k,y_k) + d_v(y_k,z_k)$$

$$\Rightarrow d_H(X,Y) + d_H(Y,Z) \geq d_v(x_k,y_k) + d_v(y_k,z_k) \quad \langle \text{Definition 3} \rangle$$

We know that $d_v(x_k,y_k) + d_v(y_k,z_k) \geq d_v(x_k,z_k)$ $\langle \text{Lemma 6} \rangle$

Then $d_H(X,Y) + d_H(Y,Z) \geq d_v(x_k,z_k)$

That is $d_H(X,Y) + d_H(Y,Z) \geq d_H(X,Z)$ $\langle \text{Assumption } d_H(X,Z) = d_v(y_k,z_k) \rangle$

2.3 Neighborhood of a Point or Pixel Using Different Distance Metrics

We define the **neighborhood** of a target point, *T*, as a set of points, *S*, such that each point in *S* is within a specified distance, *r*, from the target *T* and *S* contains all of the points that have the distance *r* or less from *T*. That is

$$X \in S \text{ if and only if } d(T, X) \leq r$$

The elements in *S* are called the **nearest neighbors** of T with respect to the distance *r* and metric *d*.

r is called the **radius** of the neighborhood.

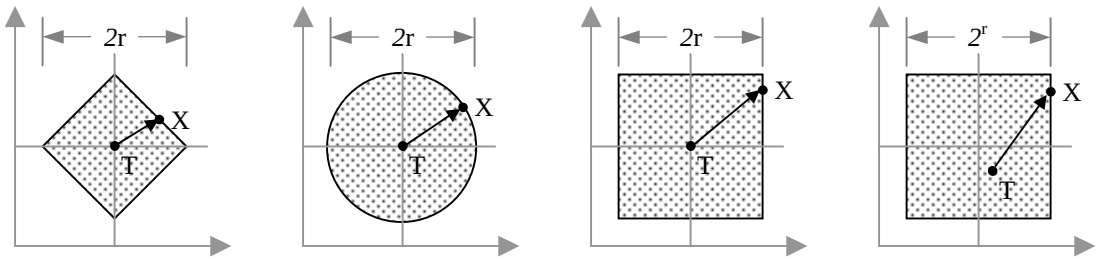
T is called the **center** of the neighborhood.

The locus of the point *X* satisfying $d(T, X) = r$, is called the **boundary** of the neighborhood.

When $r = 1$, the boundary of the neighborhood is called the **unit circle** for the metric *d*.

The neighborhood can also be defined as a closed region, *R*, in the space such that a point, *X*, is in the region *R* if and only if $d(T, X) \leq r$.

Different distance metrics result in neighborhoods with different sizes and different shapes. In a two dimensional space, the neighborhood using Manhattan distance metric is a diamond. The center of the neighborhood is the intersection point of its diagonals. Each side of the diamond makes 45° angles with the axes and the length of the diagonals is 2*r* i.e. each side is $\sqrt{2}r$. The neighborhood using Euclidian distance is a circle with radius *r* and center *T*; the center of the circle is the center of the neighborhood. Using the Max distance it is a square with sides 2*r*, center *T* and having its sides parallel to the axes. The neighborhood for HOB distance is also a square with side 2^r and the sides are parallel to the axes but the center of neighborhood, *T*, is not necessarily the center of the square. The size of the neighborhood for the Canberra and Squared Cord distance depends on the target *T*; for the same distance *r*, different targets generates different sized neighborhood. The neighborhoods using the four distance metrics discussed above are depicted in the figure 2.2.



a) *Manhattan* b) *Euclidian* c) *Max* d) *HOB*

Figure 2.2: *neighborhood using different distance metrics for 2-dimensional data points. T is the center of the neighborhood and X is a point on the boundary, i.e. d*(*T, X*) = *r. Shaded region indicates the neighborhood.*

2.4 Decision Boundaries for the Distance Metrics

Let *A* and *B* be two stationary points and *X* be a moving point in the space. The locus of the point *X* satisfying the condition $d(A, X) = d(B, X)$ is a hyperplane *D* (a line in a 2-dimensional space), which divides the space into two half-planes (shown in figure 2.3).

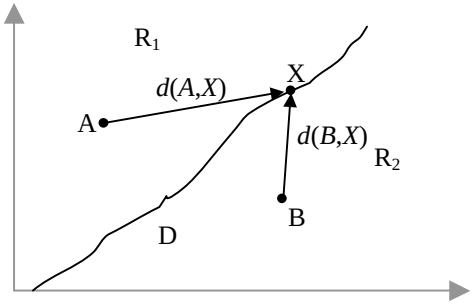


Figure 2.3: *Decision boundary between points A and B using an arbitrary distance metric d.*

The points in the region or half-plane R_1 are closer to point *A*; the points in the region R_2 are closer to the point *B* and the points on the hyperplane *D* have the same distance from *A* and *B*. This hyperplane *D* is called the decision boundary between points *A* and *B* for the metric *d*.

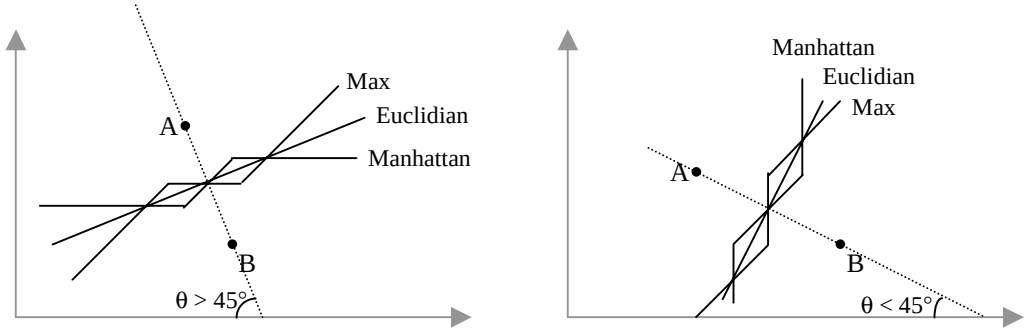


Figure 2.4: *Decision boundary for Manhattan, Euclidian and Max distance.*

The decision boundary for Euclidian distance is the perpendicular bisector to the line segment joining points *A* and *B*. The decision boundary for Manhattan distance is a 3-segment line; the middle segment is a straight line making a 45° angle with the axes and other two segments are parallel to the axes. For Max distance, it is also a 3-segment line;

the middle segment is parallel to the axes and the other two segments are straight line making a 45° angle with the axes. The parallel segment can be parallel to the *x*-axis or the *y*-axis depending on the orientation of points *A* and *B*. The decision boundary for HOB distance is a straight line perpendicular to the axis that makes the largest distance i.e. the axis, *k*, for which $d_v(x_k, y_k)$ is maximum.

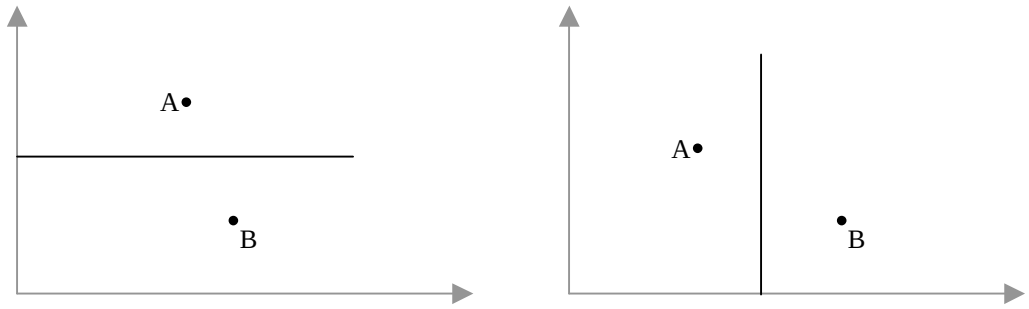


Figure 2.5: *Decision boundary for HOB distance*

CHAPTER 3: P-TREES AND ITS ALGEBRA & PROPERTIES

3.1 P-trees and Its Algebra

Spatial data is the type of data where each data tuple corresponds to a unique point in the space, usually a two dimensional space. The representation of spatial data is very important for further processing, such as data mining. Remotely Sensed Imagery (RSI) data belongs to the spatial data category. 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, TM (Thematic Mapper) scenes contain at least seven bands (Blue, Green, Red, NIR, MIR, TIR and MIR2) while a TIFF image contains three bands (Blue, Green and Red). Each band contains a relative reflectance intensity value in the range 0-to-255 (one byte) for each pixel location in the scene. 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.

Most spatial data comes in a format called BSQ for Band Sequential (or can be easily converted to BSQ). BSQ data has a separate file for each band. The ordering of the data values within a band is raster ordering with respect to the spatial area represented in the dataset. This order is assumed and therefore is not explicitly indicated as a key attribute in each band (bands have just one column). In this paper, we divided each BSQ band into several files, one for each bit position of the data values. We call this format **bit Sequential** or **bSQ** [8, 12, 13]. A Landsat Thematic Mapper satellite image, for example,

is in BSQ format with 7 bands, B1,…,B7, (Landsat-7 has 8) and ~40,000,000 8-bit data values. A typical TIFF image aerial digital photograph is in what is called Band Interleaved by Bit (BIP) format, in which there is one file containing ~24,000,000 bits ordered by their positions, then band and then raster-ordered-pixel-location. A simple transform can be used to convert TIFF images to BSQ and then to bSQ format.

We organize each bSQ bit file, Bij (the file constructed from the j^{th} bits of i^{th} band), into a tree structure, called a Peano Count Tree (P-tree). A P-tree is a quadrant-based tree. The root of a P-tree contains the 1-bit count of the entire bit-band. The next level of the tree contains the 1 bit counts of the four quadrants in Peano order or Z-order, which is shown bellow.

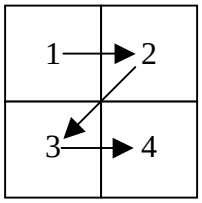
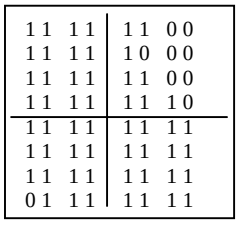
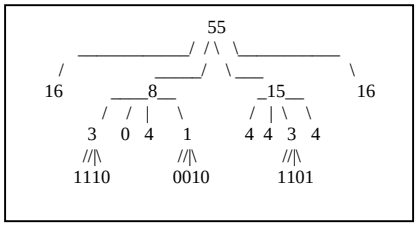


Figure 3.1 *Peano ordering or Z-ordering*

At the next level, each quadrant is partitioned into sub-quadrants and their 1-bit counts in raster order constitute the children of the quadrant node. This construction is continued recursively down each tree path until the sub-quadrant is ***pure*** (entirely 1-bits or entirely 0-bits), which may or may not be at the leaf level. For example, the P-tree for a 8-row-8-column bit-band is shown in Figure 1. If all of the bits in a quadrant are 0 (1), the corresponding node in the P-tree is called **pure0** (**pure1**) node.





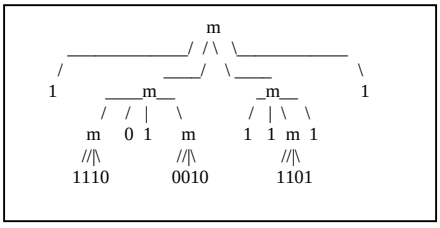


Figure 3.2. *8-by-8 image and its P-tree (P-tree and PM-tree)*

In this example, 55 is the count of 1's in the entire image, the numbers at the next level, 16, 8, 15 and 16, are the 1-bit counts for the four major quadrants. Since the first and last quadrant are made up of entirely 1-bits, we do not need sub-trees for these two quadrants. This pattern is continued recursively. Recursive raster ordering is called the Peano or Z-ordering in the literature – therefore, the name Peano Count trees. The process will definitely terminate at the "leaf" level where each quadrant is a 1-row-1-column quadrant. If we were to expand all sub-trees, including those for quadrants that are pure 1-bits, then the leaf sequence is just the Peano space-filling curve for the original raster image.

For each band (assuming 8-bit data values), we get 8 basic P-trees, one for each bit positions. For band, Bi, we will label the basic P-trees, $P_{i,1}$, $P_{i,2}$, …, $P_{i,8}$, thus, $P_{i,j}$ is a lossless representation of the j^{th} bits of the values from the i^{th} band. However, P_{ij} provides much more information and is structured to facilitate many important data mining processes.

For efficient implementation, we use a variation of basic P-trees, called a PM-tree (Pure Mask tree). In the PM-tree, we use a 3-value logic, in which 11 represents a quadrant of pure 1-bits (pure1 quadrant), 00 represents a quadrant of pure 0-bits (pure0 quadrant) and 01 represents a mixed quadrant. To simplify the exposition, we use 1 instead of 11 for pure1, 0 for pure0, and m for mixed. The PM-tree for the previous example is also given in Figure 3.2.

P-tree algebra contains operators, AND, OR, NOT and XOR, which are the pixel-by-pixel logical operations on P-trees. The NOT operation is a straightforward translation of

each count to its quadrant-complement (e.g., a 5 count for a quadrant of 16 pixels has complement of 11). The AND operations is described in full detail below. The OR is identical to the AND except that the role of the 1-bits and the 0-bits are reversed.

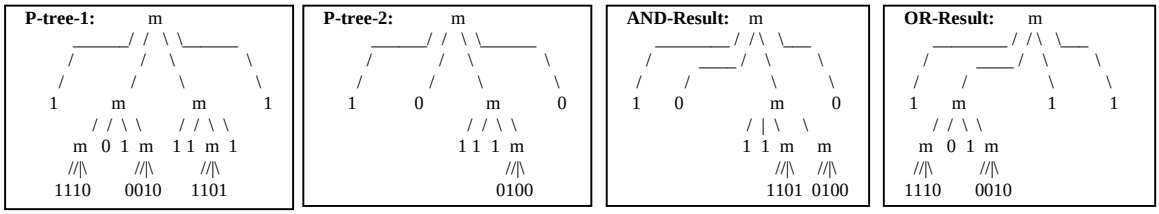


Figure 3.3: *P-tree Algebra*

The basic P-trees can be combined using simple logical operations to produce P-trees for the original values (at any level of precision, 1-bit precision, 2-bit precision, etc.). We let $P_{b,v}$ denote the Peano Count Tree for band, b, and value, v, where v can be expressed in 1-bit, 2-bit,.., or 8-bit precision. Using the full 8-bit precision (all 8 –bits) for values, $P_{b,11010011}$ can be constructed from the basic P-trees as:

$P_{b,11010011} = P_{b1}$ AND P_{b2} AND P_{b3}' AND P_{b4} AND P_{b5}' AND P_{b6}' AND P_{b7} AND $P_{b8,}$ where ' indicates NOT operation. The AND operation is simply the pixel-wise AND of the bits.

A P-tree representing the range of the values is called the **range P-tree or interval P-tree**. An interval P-tree can be contructed performing OR operations on value P-trees. Interval P-tree for the range [v1, v2] for the band 1 is

$$P_{1,(v1,v2)} = P_{1,v1} \text{ OR } P_{1,v1+1} \text{ OR } P_{1,v1+2} \text{ OR } \ldots \text{ OR } P_{1,v2}$$

3.2 Properties of P-trees

In the rest of the paper we shall use the following notations that make easier writing P-tree expressions.

$p_{x,y}$ is the pixel with coordinate *(x, y)*.

$V_{x,y,i}$ is the value for the band *i* of the pixel $p_{x,y}$.

$b_{x,y,i,j}$ is the j^{th} bit of $V_{x,y,i}$ (bits are numbered from left to right, $b_{x,y,i,0}$ is the leftmost bit).

Indices:

x: column (*x*-coordinate), *y*: row (*y*-coordinate), *i*: band, *j*: bit

For any P-trees *P*, P_1 and P_2,

P_1 & P_2 denotes P_1 *AND* P_2

$P_1 \mid P_2$ denotes P_1 *OR* P_2

$P_1 \oplus P_2$ denotes P_1 *XOR* P_2

P' denotes COMPLEMENT of *P*

Basic P-trees: $P_{band,\ bit}$

$P_{i,j}$ is the basic P-tree for bit *j* of band *i*.

Value P-trees: $P_{band}(value)$

$P_i(v)$ is the P-tree for the value *v* of band *i*.

Range P-trees: $P_{band}(lower\_limit, upper\_limit)$

$P_i(v_1, v_2)$ is the P-tree for the range $[v_1, v_2]$ of band *i*.

rc(P) is the root count, the count stored at the root node, of P-tree *P*.

P^0 is *pure*0-*tree*, a P-tree having the root node which is *pure*0.

P^1 is *pure*1-*tree*, a P-tree having the root node which is *pure*1.

N is the number of pixels in the image or space under consideration.

c is the number of rows or height of the image in pixel.

r is the number of columns or width of the image in pixel.

n is the number of bits in each band-value.

Lemma 1:

a) *For any two P-trees P_1 and P_2, $rc(P_1 \mid P_2) = 0 \Rightarrow rc(P_1) = 0$ and $rc(P_2) = 0$.*

More strictly, *$rc(P_1 \mid P_2) = 0$, if and only if $rc(P_1) = 0$ and $rc(P_2) = 0$.*

Proof: (Proof by contradiction) Let, *$rc(P_1) \neq 0$.* Then, for some pixels there are 1s in P_1 and for those pixels there must be 1s in $P_1 \mid P_2$ i.e. *$rc(P_1 \mid P_2) \neq 0$,* But we assumed *$rc(P_1 \mid P_2) = 0$.* Therefore *$rc(P_1) = 0$.* Similarly we can prove that *$rc(P_2) = 0$.*

The proof for the inverse, *$rc(P_1) = 0$ and $rc(P_2) = 0 \Rightarrow rc(P_1 \mid P_2) = 0$* is trivial.

This immediately follows the definitions.

b) *$rc(P_1) = 0$ or $rc(P_2) = 0 \Rightarrow rc(P_1 \,\&\, P_2) = 0$*

c) *$rc(P_1) = 0$ and $rc(P_2) = 0 \Rightarrow rc(P_1 \,\&\, P_2) = 0$,* in fact, this is covered by (b).

Proofs are immediate.

Lemma 2:

a) $rc(P^0) = 0$

b) $rc(P^1) = N$

c) $rc(P) = 0 \Leftrightarrow P = P^0$

d) $rc(P) = N \Leftrightarrow P = P^1$

e) $P \,\&\, P^0 = P^0$

f) $P \,\&\, P^1 = P$

g) $P \mid P^0 = P$

h) $P \mid P^1 = P^1$

i) $P \,\&\, P' = P^0$

j) $P \mid P' = P^1$

k) $P \oplus P^1 = P'$

l) $P \oplus P^0 = P$

m) $P \oplus P' = P^1$

n) $P \,\&\, P = P$

n) $P \mid P = P$

n) $P \oplus P = P'$

Proofs are immediate.

Lemma 3: $v_1 \neq v_2 \Rightarrow rc\{P_i(v_1) \,\&\, P_i(v_2)\} = 0$, *for any band i.*

Proof: $P_i(v)$ represents all the pixels having value v for the band i. If $v_1 \neq v_2$, no pixel can have the values both v_1 and v_2 for the same band. Therefore, if there is a 1 in $P_i(v_1)$ for any pixel, there must be 0 in $P_i(v_2)$ for that pixel and vice versa. Hence $rc\{P_i(v_1) \,\&\, P_i(v_2)\} = 0$.

Lemma 4: $rc(P_1 \mid P_2) = rc(P_1) + rc(P_2) - rc(P_1 \,\&\, P_2)$.

Proof: Let

the number of pixels for which there are 1s in P_1 and 0s in P_2 is n_1,

the number of pixels for which there are 0s in P_1 and 1s in P_2 is n_2

and the number of pixels for which there are 1s in both P_1 and P_2 is n_3.

Now, $rc(P_1) = n_1 + n_3$,

$rc(P_2) = n_2 + n_3$,

$rc(P_1 \,\&\, P_2) = n_3$

and $rc(P_1 \mid P_2) = n_1 + n_2 + n_3$

$= (n_1 + n_3) + (n_2 + n_3) - n_3$

$= rc(P_1) + rc(P_2) - rc(P_1 \,\&\, P_2)$

Theorem 1: $rc\{P_i(v_1) \mid P_i(v_2)\} = rc\{P_i(v_1)\} + rc\{P_i(v_2)\}$, where $v_1 \neq v_2$.

Proof: $rc\{P_i(v_1) \mid P_i(v_2)\} = rc\{P_i(v_1)\} + rc\{P_i(v_2)\} - rc\{P_i(v_1) \,\&\, P_i(v_2)\}$ (Lemma 4)

If $v_1 \neq v_2$, $rc\{P_i(v_1) \,\&\, P_i(v_2)\} = 0$. (Lemma 3)

Therefore, $rc\{P_i(v_1) \mid P_i(v_2)\} = rc\{P_i(v_1)\} + rc\{P_i(v_2)\}$.

3.3 Header of a P-tree file

To make a generalized P-tree structure the following header for a P-tree file is proposed.

1 word	2 word	2 words	4 words	4 words	
Format Code	Fan-out	# of levels	Root count	Length of the body in bytes	Body of the P-tree

Figure 3.4: *Header of a P-tree file.*

Format code: Format code identifies the format of the P-tree, whether it is a *PCT* or *PMT* or in any other format. Although it is possible to recognize the format from the extension of the file, using format code is a good practice because some other applications may use the same extension for their purpose. Therefore to make sure that it is a P-tree file with the specific format we need format code. Moreover, in any standard file format such as PDF and TIFF, a file identification code is used along with the specified file extension. We propose the following codes for the P-tree formats.

0707 H[*] – PCT	1717 H – PMT	2727 H – PVT	3737 H – P0T
4747 H – P1T	5757 H – P0V	6767 H – P1V	7777 H – PNZV

Fan-out: This field contains the fan-out information of the P-tree. Fan-out information is required to traverse the P-tree in performing various P-tree operations including *AND*, *OR* and *Complement.*

# of levels: Number of levels in the P-tree. When we encounter a *pure1* or *pure0* node, we cannot tell whether it is an interior node or a leaf unless we know the level of that node and

[*] H stands for hexadecimal

the total number of levels of the tree. This is also required to know the number of 1s represented by a *pure1* node.

Root count: *Root count* i.e. the number 1s in the P-tree. Though we can calculate the *root count* of a P-tree on the fly from the P-tree data, these only 4 bytes of space can save computation time when we don't need to perform any AND/OR operations and need the root count of an existing P-tree such as the basic P-trees. The root count of a P-tree can be computed at the time of construction of the P-tree with a very little extra cost.

Length of the body: Length of the body is the size of the P-tree file in bytes excluding the header. Sometimes we may want to load the whole P-tree into RAM to increase the efficiency of computation. Since the sizes of the P-trees vary, we need to allocate memory dynamically and know the size of the required memory prior to read from disk.

3.4 Dealing with the Padded Zeros

We measure the height and the width of the images in pixels. To construct P-trees, the image must be square i.e. height and width must be equal and must be power of 2. For example an image size can be 256×256 or 512×512. Zeros are padded to the right and bottom of the image to convert it into the required size. Also a missing value can be replaced with zero. To deal with these inserted or padded zeros we need to make corrections to the root count of the final P-tree expression before using it.

Solution 1:

Every P-tree expression is a function of the basic P-trees.

Let, $P_{exp} = f_p(P_1, P_2, P_3, \ldots, P_n)$,

where P_i is a basic P-tree for $i = 1, 2, 3, \ldots, n$.

Transform the P-tree expression, P_{exp} into a Boolean expression by replacing the basic P-trees with 0 and considering the P-tree operators as a corresponding Boolean operator.

Boolean expression $B_{exp} = f_b(0, 0, 0, \ldots, 0)$.

Where f_b is the corresponding Boolean function to P-tree function f_p.

For Example, $P_{exp} = (P_1 \,\&\, P_2)' \mid P_3$

then, $B_{exp} = (0 \,\&\, 0)' \mid 0$

$= 0' \mid 0 = 1 \mid 0 = 1$

Now if $B_{exp} = 1$, corrected root count = $rc(P_{exp}) - M$,

where M is the number of padded zeros and missing values.

otherwise, no correction is necessary, i.e. corrected root count = $rc(P_{exp})$

Solution 2:

Another solution to find the corrected root count is to use a **mask** or **template P-tree**, P_t, which is formed by using a 1 bit for the existing pixels and 0 bit for the padded zeros and missing values.

Then the corrected root count = $rc(P_{exp} \,\&\, P_t)$

CHAPTER 4: PAPER 1

K-NEAREST NEIGHBOR CLASSIFICATION ON SPATIAL DATA STREAMS USING P-TREES

Abstract

In this paper we consider the classification of spatial data streams, where the training dataset changes often. New training data arrive continuously and are added to the training set. For these types of data streams, building a new classifier each time can be very costly with most techniques. In this situation, k-nearest neighbor (KNN) classification is a very good choice, since no residual classifier needs to be built ahead of time. For that reason KNN is called a lazy classifier. KNN is extremely simple to implement and lends itself to a wide variety of variations. The traditional k-nearest neighbor classifier finds the k nearest neighbors based on some distance metric by finding the distance of the target data point from the training dataset, then finding the class from those nearest neighbors by some voting mechanism. There is a problem associated with KNN classifiers. They increase the classification time significantly relative to other non-lazy methods. To overcome this problem, in this paper we propose a new method of KNN classification for spatial data streams using a new, rich, data-mining-ready structure, the Peano-count-tree or P-tree. In our method, we merely perform some logical AND/OR operations on P-trees to find the nearest neighbor set of a new sample and assign the class label. We have fast and efficient algorithms for AND/OR operations on P-trees, which reduce the classification time significantly, compared with traditional KNN classifiers. Instead of taking exactly the k nearest neighbors we form a closed-KNN set. Our experimental results show closed-KNN yields higher classification accuracy as well as significantly higher speed.

Keywords: Data Mining, K-Nearest Neighbor Classification, P-tree, Spatial Data, Data Streams.

4.1 Introduction

Classification is the process of finding a set of models or functions that describes and distinguishes data classes or concepts for the purpose of predicting the class of objects whose class labels are unknown [9]. The derived model is based on the analysis of a set of training data whose class labels are known. Consider each training sample has n attributes: A_1, A_2, A_3, …, A_{n-1}, C, where C is the class attribute which defines the class or category of the sample. The model associates the class attribute, C, with the other attributes. Now consider a new tuple or data sample whose values for the attributes A_1, A_2, A_3, …, A_{n-1} are known, while for the class attribute is unknown. The model predicts the class label of the new tuple using the values of the attributes A_1, A_2, A_3, …, A_{n-1}.

There are various techniques for classification such as Decision Tree Induction, Bayesian Classification, and Neural Networks [9, 11]. Unlike other common classification methods, a k-nearest neighbor classification (**KNN classification**) does not build a classifier in advance. That is what makes it suitable for data streams. When a new sample arrives, KNN finds the k neighbors nearest to the new sample from the training space based on some suitable similarity or closeness metric [3, 7, 10]. A common similarity function is based on the Euclidian distance between two data tuples [3]. For two tuples, $X = \langle x_1, x_2, x_3, \ldots, x_{n-1} \rangle$ and $Y = \langle y_1, y_2, y_3, \ldots, y_{n-1} \rangle$ (excluding the class labels), the **Euclidian** distance function is $d_2(X,Y) = \sqrt{\sum_{i=1}^{n-1} (x_i - y_i)^2}$. A generalization of the Euclidean function is the **Minkowski** distance function is

$d_q(X,Y) = \sqrt[q]{\sum_{i=1}^{n-1} w_i |x_i - y_i|^q}$. The Euclidean function results by setting q to 2 and each weight, w_i, to 1. The **Manhattan** distance, $d_1(X,Y) = \sum_{i=1}^{n-1} |x_i - y_i|$ result by setting q to 1. Setting q to ∞, results in the **max** function $d_\infty(X,Y) = \max_{i=1}^{n-1} |x_i - y_i|$. After finding the k nearest tuples based on the selected distance metric, the plurality class label of those k tuples can be assigned to the new sample as its class. If there is more than one class label in plurality, one of them can be chosen arbitrarily.

In this paper, we also used our new distance metric called Higher Order Bit or HOB distance and evaluated the effect of all of the above distance metrics in classification time and accuracy. HOB distance provides an efficient way of computing neighborhood while keeping the classification accuracy very high. The details of the distance metrics have been discussed in chapter 2.

Nearly every other classification model trains and tests a residual "classifier" first and then uses it on new samples. KNN does not build a residual classifier, but instead, searches again for the k-nearest neighbor set for each new sample. This approach is simple and can be very accurate. It can also be slow (the search may take a long time). KNN is a good choice when simplicity and accuracy are the predominant issues. KNN can be superior when a residual, trained and tested classifier has a short useful lifespan, such as in the case with data streams, where new data arrives rapidly and the training set is ever changing [1, 2]. For example, in spatial data, AVHRR images are generated in every one hour and can be viewed as spatial data streams. The purpose of this paper is to introduce a new KNN-like model, which is not only simple and accurate but is also fast – fast enough for use in spatial data stream classification.

In this paper we propose a simple and fast KNN-like classification algorithm for spatial data using P-trees. P-trees are new, compact, data-mining-ready data structures, which provide a lossless representation of the original spatial data [8, 12, 13]. We consider a space to be represented by a 2-dimensional array of locations (though the dimension could just as well be 1 or 3 or higher). Associated with each location are various attributes, called ***bands***, such as visible reflectance intensities (blue, green and red), infrared reflectance intensities (e.g., NIR, MIR1, MIR2 and TIR) and possibly other value bands (e.g., crop yield quantities, crop quality measures, soil attributes and radar reflectance intensities). One band such as the yield band can be the class attribute. The location coordinates in raster order constitute the key attribute of the spatial dataset and the other bands are the non-key attributes. We refer to a location as a pixel in this paper.

Using P-trees, we presented two algorithms, one based on the **max** distance metric and the other based on our new **HOBS** distance metric. **HOBS** is the similarity of the most significant bit positions in each band. It differs from pure Euclidean similarity in that it can be an asymmetric function depending upon the bit arrangement of the values involved. However, it is very fast, very simple and quite accurate. Instead of using exactly k nearest neighbor (a KNN set), our algorithms build a **closed-KNN** set and perform voting on this closed-KNN set to find the predicting class. Closed-KNN, a superset of KNN, is formed by including the pixels, which have the same distance from the target pixel as some of the pixels in KNN set. Based on this similarity measure, finding nearest neighbors of new samples (pixel to be classified) can be done easily and very efficiently using P-trees and we found higher classification accuracy than traditional methods on considered datasets. The classification algorithms to find nearest

neighbors have been given in the section 4.2. We provided the experimental results and analyses in section 4.3. And section 4.4 is the conclusion.

4.2 Classification Algorithm

In the original k-nearest neighbor (KNN) classification method, no classifier model is built in advance. KNN refers back to the raw training data in the classification of each new sample. Therefore, one can say that the entire training set is the classifier. The basic idea is that the similar tuples most likely belongs to the same class (a continuity assumption). Based on some pre-selected distance metric (some commonly used distance metrics are discussed in introduction), it finds the k most similar or nearest training samples of the sample to be classified and assign the plurality class of those k samples to the new sample. The value for k is pre-selected. Using relatively larger k may include some pixels that are not so similar to the target pixel and on the other hand, using very smaller k may exclude some potential candidate pixels. In both cases the classification accuracy will decrease. The optimal value of k depends on the size and nature of the data. The typical value for k is 3, 5 or 7. The steps of the classification process are:

1) Determine a suitable distance metric.
2) Find the k nearest neighbors using the selected distance metric.
3) Find the plurality class of the k-nearest neighbors (voting on the class labels of the NNs).
4) Assign that class to the sample to be classified.

We provided two different algorithms using P-trees, based on two different distance metrics **max** (Minkowski distance with $q = \infty$) and our newly defined **HOB** distance. Instead of examining individual pixels to find the nearest neighbors, we start our initial neighborhood (neighborhood is a set of neighbors of the target pixel within a

specified distance based on some distance metric, not the spatial neighbors, neighbors with respect to values) with the target sample and then successively expand the neighborhood area until there are k pixels in the neighborhood set. The expansion is done in such a way that the neighborhood always contains the closest or most similar pixels of the target sample. The different expansion mechanisms implement different distance functions. In the next section (section 3.1) we described the distance metrics and expansion mechanisms.

Of course, there may be more boundary neighbors equidistant from the sample than are necessary to complete the k nearest neighbor set, in which case, one can either use the larger set or arbitrarily ignore some of them. To find the exact k nearest neighbors one has to arbitrarily ignore some of them.

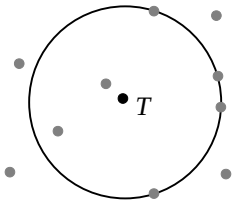


Figure 4.1: *Closed-KNN set.*
T, the pixel in the center is the target pixels. With k = 3, to find the third nearest neighbor, we have four pixels (on the boundary line of the neighborhood) which are equidistant from the target.

Instead we propose a new approach of building nearest neighbor (NN) set, where we take the closure of the k-NN set, that is, we include all of the boundary neighbors and we call it the **closed-KNN** set. Obviously closed-KNN is a superset of KNN set. In the above example, with k = 3, KNN includes the two points inside the circle and any one point on the boundary. The closed-KNN includes the two points in side the circle and all of the four boundary points. The inductive definition of the closed-KNN set is given below.

***Definition* 4.1:** a) *if* $x \in KNN$, *then* $x \in$ *closed-KNN*

b) *if* $x \in$ *closed-KNN and* $d(T,y) \leq d(T,x)$, *then* $y \in$ *closed-KNN*

Where, d(T,x) is the distance of x from target T.

c) *closed-KNN does not contain any pixel, which cannot be produced by step a and b.*

Our experimental results show closed-KNN yields higher classification accuracy than KNN does. The reason is if for some target there are many pixels on the boundary, they have more influence on the target pixel. While all of them are in the nearest neighborhood area, inclusion of one or two of them does not provide the necessary weight in the voting mechanism. One may argue that then why don't we use a higher k? For example using k = 5 instead of k = 3. The answer is if there are too few points (for example only one or two points) on the boundary to make k neighbors in the neighborhood, we have to expand neighborhood and include some not so similar points which will decrease the classification accuracy. We construct closed-KNN only by including those pixels, which are in as same distance as some other pixels in the neighborhood without further expanding the neighborhood. To perform our experiments, we find the optimal k (by trial and error method) for that particular dataset and then using the optimal k, we performed both KNN and closed-KNN and found higher accuracy for P-tree-based closed-KNN method. The experimental results are given in section 4. In our P-tree implementation, no extra computation is required to find the closed-KNN. Our expansion mechanism of nearest neighborhood automatically includes the points on the boundary of the neighborhood.

Also, there may be more than one class in plurality (if there is a tie in voting), in which case one can arbitrarily chose one of the plurality classes. Unlike the traditional k-nearest neighbor classifier our classification method doesn't store and use raw training data. Instead we use the data-mining-ready P-tree structure, which can be built very quickly from the training data. Without storing the raw data we create the basic P-trees and store them for future classification purpose. Avoiding the examination of

individual data points and being ready for data mining these P-trees not only saves classification time but also saves storage space, since data is stored in compressed form. This compression technique also increases the speed of ANDing and other operations on P-trees tremendously, since operations can be performed on the pure0 and pure1 quadrants without reference to individual bits, since all of the bits in those quadrant are the same.

4.2.1 Expansion of Neighborhood

Similarity and distance can be measured by each other; more distance less similar and less distance more similar. Our similarity metric is the closeness in numerical values for corresponding bands. We begin searching for nearest neighbors by finding the exact matches i.e. the pixels having as same band-values as that of the target pixel. If the number of exact matches is less than k, we expand the neighborhood. For example, for a particular band, if the target pixel has the value *a*, we expand the neighborhood to the range [*a-b, a+c*], where *b* and *c* are positive integers and find the pixels having the band value in the range [*a-b, a+c*]. We expand the neighbor in each band (or dimension) simultaneously. We continue expanding the neighborhood until the number pixels in the neighborhood is greater than or equal to k. We develop the following two different mechanisms, corresponding to max distance (Minqowski distance with q = ∞ or L_∞) and our newly defined HOBS distance, for expanding the neighborhood. The two given mechanisms have trade off between execution time and classification accuracy.

A. Higher Order Bit Similarity Method (Using HOB Distance):

The HOB distance between two pixels X and Y is defined by

$$d_H(X,Y) = \max_{i=1}^{n-1}\{m - \mathrm{HOB}(x_i,y_i)\}$$

n is the total number of bands where one of them (the last band) is class attribute that we don't use for measuring similarity.

m is the number of bits in binary representations of the values. All values must be represented using the same number of bits.

$HOB(A, B) = \max\{s \mid i \leq s \Rightarrow a_i = b_i\}$

a_i and b_i are the i^{th} bits of *A* and *B* respectively.

The detailed definition of HOB distance and its behavior have been discussed in chapter 2.

To find the Closed-KNN set, first we look for the pixels, which are identical to the target pixel in all 8 bits of all bands i.e. the pixels, *X*, having distance from the target *T*, $d_p(X,T) = 0$. If, for instance, x_1=105 ($01101001_b = 105_d$) is the target pixel, the initial neighborhood is [105, 105] ([01101001, 01101001]). If the number of matches is less than k, we look for the pixels, which are identical in the 7 most significant bits, not caring about the 8^{th} bit, i.e. pixels having $d_p(X,T) \leq 1$. Therefore our expanded neighborhood is [104,105] ([01101000, 01101001] or [0110100-, 0110100-] - don't care about the 8^{th} bit). Removing one more bit from the right, the neighborhood is [104, 107] ([011010--, 011010--] - don't care about the 7^{th} or the 8^{th} bit). Continuing to remove bits from the right we get intervals, [104, 111], then [96, 111] and so on. Computationally this method is very cheap (since the counts are just the root counts of individual P-trees, all of which can be constructed in one operation). However, the expansion does not occur evenly on both sides of the target value (note: the center of the neighborhood [104, 111] is (104 + 111) /2 = 107.5 but the target value is 105). Another observation is that the size of the neighborhood is expanded by powers of 2. These uneven and jump expansions include some not so similar pixels in the neighborhood keeping the classification accuracy lower. But P-tree-based closed-KNN

method using this HOBS metric still outperforms KNN methods using any distance metric as well as becomes the fastest among all of these methods.

To improve accuracy further we propose another method called perfect centering to avoid the uneven and jump expansion. Although, in terms of accuracy, perfect centering outperforms HOBS, in terms of computational speed it is slower than HOBS.

B. Perfect Centering (using Max distance): In this method we expand the neighborhood by 1 on both the left and right side of the range keeping the target value always precisely in the center of the neighborhood range. We begin with finding the exact matches as we did in HOBS method. The initial neighborhood is [*a*, *a*], where *a* is the target band value. If the number of matches is less than k we expand it to [*a*-1, *a*+1], next expansion to [*a*-2, *a*+2], then to [*a*-3, *a*+3] and so on.

Perfect centering expands neighborhood based on max distance metric or L_∞ metric, Minkowski distance (discussed in introduction) metric setting q = ∞.

$$d_\infty(X,Y) = \max_{i=1}^{n-1} |x_i - y_i|$$

In the initial neighborhood $d_\infty(X,T)$ is 0, the distance of any pixel *X* in the neighborhood from the target *T*. In the first expanded neighborhood [*a*-1, *a*+1], $d_\infty(X,T) \leq 1$. In each expansion $d_\infty(X,T)$ increases by 1. As distance is the direct difference of the values, increasing distance by one also increases the difference of values by 1 evenly in both side of the range without any jumping.

This method is computationally a little more costly because we need to find matches for each value in the neighborhood range and then accumulate those matches but it results better nearest neighbor sets and yields better classification accuracy. We compare these two techniques later in section 4.3.

4.2.2 Computing the Nearest Neighbors

For HOBS: We have the basic P-trees of all bits of all bands constructed from the training dataset and the new sample to be classified. Suppose, including the class band, there are n bands or attributes in the training dataset and each attribute is m bits long. In the target sample we have *n*-1 bands, but the class band value is unknown. Our goal is to predict the class band value for the target sample.

$P_{i,j}$ is the P-tree for bit j of band i. This P-tree stores all the j^{th} bits of the i^{th} band of all the training pixels. The root count of a P-tree is the total counts of one bits stored in it. Therefore, the root count of $P_{i,j}$ is the number of pixels in the training dataset having a 1 value in the j^{th} bit of the i^{th} band. $P'_{i,j}$ is the complement P-tree of $P_{i,j}$. $P'_{i,j}$ stores 1 for the pixels having a 0 value in the j^{th} bit of the i^{th} band and stores 0 for the pixels having a 1 value in the j^{th} bit of the i^{th} band. Therefore, the root count of $P'_{i,j}$ is the number of pixels in the training dataset having 0 value in the j^{th} bit of the i^{th} band.

Now let, $b_{i,j} = j^{th}$ bit of the i^{th} band of the target pixel.

Define $Pt_{i,j} = P_{i,j}$, *if* $b_{i,j} = 1$

$= P'_{i,j}$, otherwise

We can say that the root count of $Pt_{i,j}$ is the number of pixels in the training dataset having as same value as the j^{th} bit of the i^{th} band of the target pixel.

Let, $Pv_{i,1\text{-}j} = Pt_{i,1}$ & $Pt_{i,2}$ & $Pt_{i,3}$ & … & $Pt_{i,j}$, here & is the P-tree AND operator.

$Pv_{i,1\text{-}j}$ counts the pixels having as same bit values as the target pixel in the higher order j bits of i^{th} band.

Using higher order bit similarity, first we find the P-tree $Pnn = Pv_{1,1\text{-}8}$ & $Pv_{2,1\text{-}8}$ & $Pv_{3,1\text{-}8}$ & … & $Pv_{n\text{-}1,1\text{-}8}$, where *n*-1 is the number of bands excluding the class band. *Pnn* represents the pixels that exactly match the target pixel. If the root count of *Pnn* is less than *k* we look for higher order 7 bits matching i.e. we calculate $Pnn = Pv_{1,1\text{-}7}$ & $Pv_{2,1\text{-}7}$

& $Pv_{3,1\text{-}7}$ & … & $Pv_{n\text{-}1,1\text{-}7}$. Then we look for higher order 6 bits matching and so on. We continue as long as root count of Pnn is less than *k*. *Pnn* represents closed-KNN set i.e. the training pixels having the as same bits in corresponding higher order bits as that in target pixel and the root count of *Pnn* is the number of such pixels, the nearest pixels. A 1 bit in Pnn for a pixel means that pixel is in closed-KNN set and a 0 bit means the pixel is not in the closed-KNN set. The algorithm for finding nearest neighbors is given in figure 4.2

Algorithm: **Finding the P-tree representing closed-KNN set using HOBS**

Input: $P_{i,j}$ for all *i* and *j*, basic P-trees of all the bits of all bands of the training dataset and $b_{i,j}$ for all *i* and *j*, the bits for the target pixels

Output: Pnn, the P-tree representing the nearest neighbors of the target pixel

// *n* is the number of bands where n^{th} band is the class band
// *m* is the number of bits in each band

FOR $i = 1$ *TO* $n\text{-}1$ *DO*
  FOR $j = 1$ *TO* m *DO*
    IF $b_{i,j} = 1$ $Pt_{ij} \leftarrow P_{i,j}$
    ELSE $Pt_{i,j} \leftarrow P'_{i,j}$

FOR $i = 1$ *TO* $n\text{-}1$ *DO*
  $Pv_{i,1} \leftarrow Pt_{i,1}$
  FOR $j = 2$ *TO* m *DO*
    $Pv_{i,j} \leftarrow Pv_{i,j-1}$ & $Pt_{i,j}$

$s \leftarrow m$ // first we check matching in all *m* bits
REPEAT
  $Pnn \leftarrow Pv_{1,s}$
  FOR $r = 2$ *TO* $n\text{-}1$ *DO*
    $Pnn \leftarrow Pnn$ & $Pv_{r,s}$
  $s \leftarrow s - 1$
UNTIL RootCount(Pnn) $\geq k$

Figure 4.2: *Algorithm to find closed-KNN set based on HOB metric.*

For Perfect Centering: Let v_i be the value of the target pixels for band *i*. $P_i(v_i)$ is the value P-tree for the value v_i in band i. $P_i(v_i)$ represents the pixels having value v_i in band *i*. For finding the initial nearest neighbors (the exact matches) using perfect centering we find $P_i(v_i)$ for all *i*. The ANDed result of these value P-trees i.e. $Pnn = P_1(v_1)$ & $P_2(v_2)$ & $P_3(v_3)$ & … & $P_{n\text{-}1}(v_{n\text{-}1})$ represents the pixels having the same values in each band as that of the target pixel. A value P-tree, $P_i(v_i)$, can be computed by finding the P-tree representing the pixels having the same bits in band *i* as the bits in value v_i. That is, if $Pt_{i,j} = P_{i,j}$, when $b_{i,j} = 1$ and $Pt_{i,j} = P'_{i,j}$, when $b_{i,j} = 0$ ($b_{i,j}$ is the j^{th} bit of value v_i),then

$P_i(v_i) = Pt_{i,1}$ & $Pt_{i,2}$ & $Pt_{i,3}$ & … & $Pt_{i,m}$, m is the number of bits in a band. The algorithm for computing value P-trees is given in figure 4.3 (b).

Algorithm: **Finding the P-tree representing closed-KNN set using max distance metric (perfect centering)**
Input: $P_{i,j}$ for all i and j, basic P-trees of all the bits of all bands of the training dataset and v_i for all i, the band values for the target pixel
Output: Pnn, the P-tree representing the closed-KNN set
// n is the number of bands where n^{th} band is the class band
// m is the number of bits in each band

FOR i = 1 *TO n*-1 *DO*
    $Pr_i \leftarrow P_i(v_i)$
$Pnn \leftarrow Pr_1$
FOR i = 2 *TO n*-1 *DO*
    $Pnn \leftarrow Pnn \,\&\, Pr_i$ // the initial neighborhood for exact matching
d ← 1 // distance for the first expansion
WHILE RootCount(*Pnn*) < *k DO*
    FOR i = 1 *to n*-1 *DO*
        $Pr_i \leftarrow Pr_i \mid P_i(v_i\text{-}d) \mid P_i(v_i\text{+}d)$ // neighborhood expansion
    $Pnn \leftarrow Pr_1$ // '|' is the P-tree OR operator
    FOR i = 2 *TO n*-1 *DO*
        Pnn ← *Pnn* AND Pr_i // updating closed-KNN set
    d ← d + 1

Algorithm: **Finding value P-tree**

Input: $P_{i,j}$ for all j, basic P-trees of all the bits of band i and the value v_i for band i.
Output: $P_i(v_i)$, the value p-tree for the value v_i
// m is the number of bits in each band
// $b_{i,j}$ is the j^{th} bit of value v_i

FOR j = 1 *TO m DO*
    IF $b_{i,j}$ = 1 $Pt_{ij} \leftarrow P_{i,j}$
    ELSE $Pt_{i,j} \leftarrow P'_{i,j}$

$P_i(v) \leftarrow Pt_{i,1}$
FOR j = 2 *TO m DO*
    $P_i(v) \leftarrow P_i(v)$ & $Pt_{i,j}$

Figure 4.3(a): *Algorithm to find closed-KNN set based on Max metric (Perfect Centering).*

4.3(b): *Algorithm to compute value P-trees*

If the number of exact matching i.e. root count of *Pnn* is less than k, we expand neighborhood along each dimension. For each band i, we calculate range P-tree $Pr_i = P_i(v_i\text{-}1) \mid P_i(v_i) \mid P_i(v_i\text{+}1)$. '|' is the P-tree OR operator. Pr_i represents the pixels having a value either v_i-1 or v_i or v_i+1 i.e. any value in the range [v_i-1, v_i+1] of band i. The *AND*ed result of these range P-trees, Pr_i for all i, produce the expanded neighborhood, the pixels having band values in the ranges of the corresponding bands. We continue this expansion process until root count of *Pnn* is greater than or equal to k. The algorithm is given in figure 4.3(a).

4.2.3 Finding the plurality class among the nearest neighbors

For the classification purpose, we don't need to consider all bits in the class band. If the class band is 8 bits long, there are 256 possible classes. Instead of considering 256 classes we partition the class band values into fewer groups by considering fewer significant bits. For example if we want to partition into 8 groups we can do it by

truncating the 5 least significant bits and keeping the most significant 3 bits. The 8 classes are 0, 1, 2, 3, 4, 5, 6 and 7. Using these 3 bits we construct the value P-trees $P_n(0)$, $P_n(1)$, $P_n(2)$, $P_n(3)$, $P_n(4)$, $P_n(5)$, $P_n(6)$, and $P_n(7)$.

An 1 value in the nearest neighbor P-tree, *Pnn*, indicates that the corresponding pixel is in the nearest neighbor set. An 1 value in the value P-tree, $P_n(i)$, indicates that the corresponding pixel has the class value *i*. Therefore $Pnn \ \& \ P_n(i)$ represents the pixels having a class value *i* and are in the nearest neighbor set. An *i* which yields the maximum root count of Pnn & $P_n(i)$ is the plurality class. The algorithm is given in figure 4.4.

Algorithm: **Finding the plurality class**

Input: $P_n(i)$, the value P-trees for all class *i* and the closed-KNN P-tree, *Pnn*
Output: the plurality class
// *c* is the number of different classes

```
class ← 0
P ← Pnn & Pn(0)
rc ← RootCount(P)
FOR i = 1 TO c - 1 DO
    P ← Pnn & Pn(i)
    IF rc < RootCount(P)
        rc ← RootCount(P)
        class ← i
```

Figure 4.4: *Algorithm to find the plurality class*

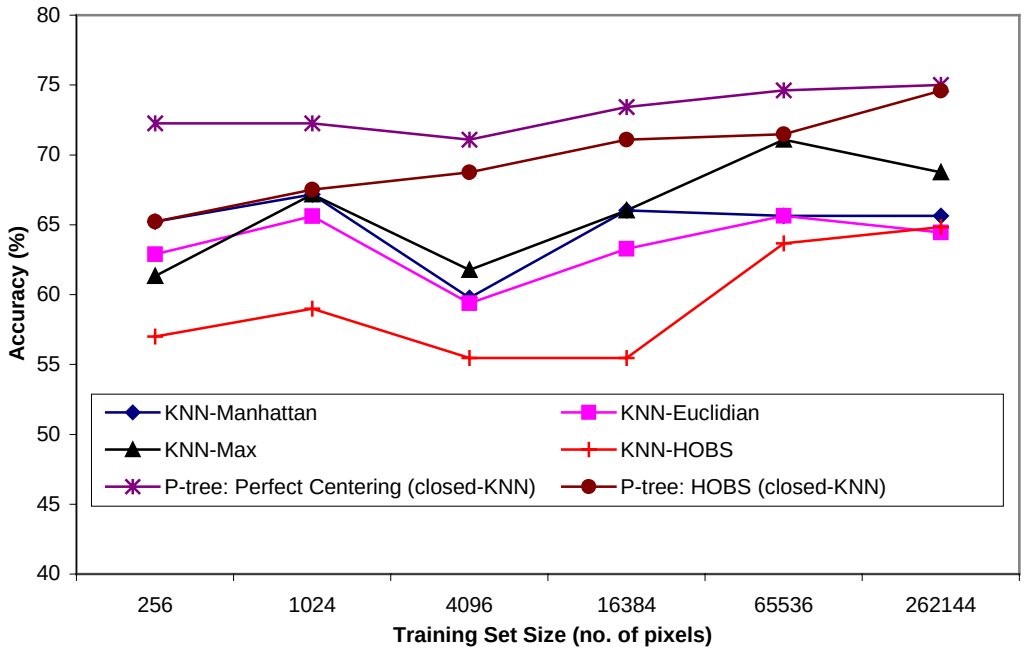
4.3 Performance Analysis

We performed experiments on two sets of Arial photographs of the Best Management Plot (BMP) of Oakes Irrigation Test Area (OITA) near Oaks, North Dakota, United States. The latitude and longitude are 45°49’15”N and 97°42’18”W respectively. The two images “29NW083097.tiff” and “29NW082598.tiff” have been taken in 1997 and 1998 respectively. Each image contains 3 bands, red, green and blue reflectance values. Three other separate files contain synchronized soil moisture, nitrate and yield values. Soil moisture and nitrate are measured using shallow and deep well

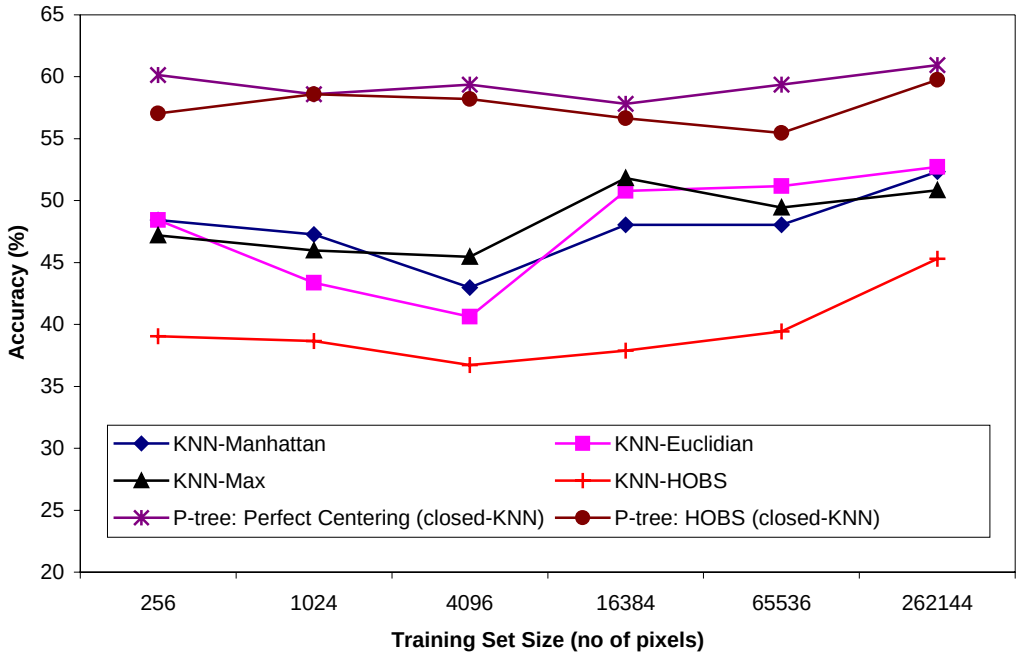
lysimeters. Yield values were collected by using a GPS yield monitor on the harvesting equipments. The datasets are available at http://datasurg.ndsu.edu/.

Among those 6 bands we consider the yield as class attribute. Each band is 8 bits long. So we have 8 basic P-trees for each band and 40 (for the other 5 bands except yield) in total. For the class band, yield, we considered only the most significant 3 bits. Therefore we have 8 different class labels for the pixels. We built 8 value P-trees from the yield values – one for each class label.

The original image size is 1320×1320. For experimental purpose we form 16×16, 32×32, 64×64, 128×128, 256×256 and 512×512 image by choosing pixels that are uniformly distributed in the original image. In each case, we form one test set and one training set of equal size. For each of the above sizes we tested KNN with Manhattan, Euclidian, Max and HOBS distance metrics and our two P-tree methods, Perfect Centering and HOBS. The accuracies of these different implementations are given in the figure 4.5 for both of the datasets.



(a) *29NW083097.tiff and associated other files (*1997 *dataset)*



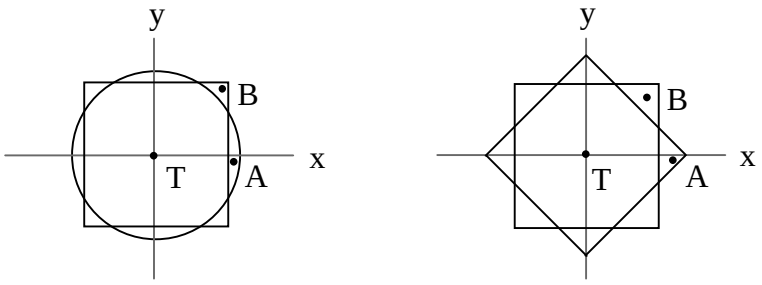
(b) *29NW082598.tiff and associated other files (*1998 *dataset)*

Figure 4.5: *Accuracy of different implementations for the* 1997 *and* 1998 *datasets*

We see that both of our P-tree based closed-KNN methods outperform the KNN methods for both of the datasets. The reasons are discussed in section 3. We discussed in section 3.1, why the perfect centering methods performs better than HOBS. We also implemented the HOBS metric for KNN standard. From the result we can see that the accuracy is very poor. The HOBS metric is not suitable for a KNN approach since HOBS does not provide a neighborhood with the target pixel in the exact center. Increased accuracy of HOBS in P-tree implementation is the effect of closed-KNN, which is explained in section 3. In a P-tree implementation, the ease of computability for closed-KNN using HOBS makes it a superior method. The P-tree based HOBS is the fastest method where as the KNN-HOBS is still the poorest (figure 4.7).

Another observation is that for 1997 data (Figure 4.5(a)), in KNN implementations, the max metric performs much better than other three metrics. For the 1998 dataset, max is competitive with other three metrics. In many cases, as well as for image data, max metrics can be the best choice. In our P-tree implementations, we also get very

high accuracy with the max distance (perfect centering method). We can understand this by examining the shape of the neighborhood for different metrics (figure 4.6).



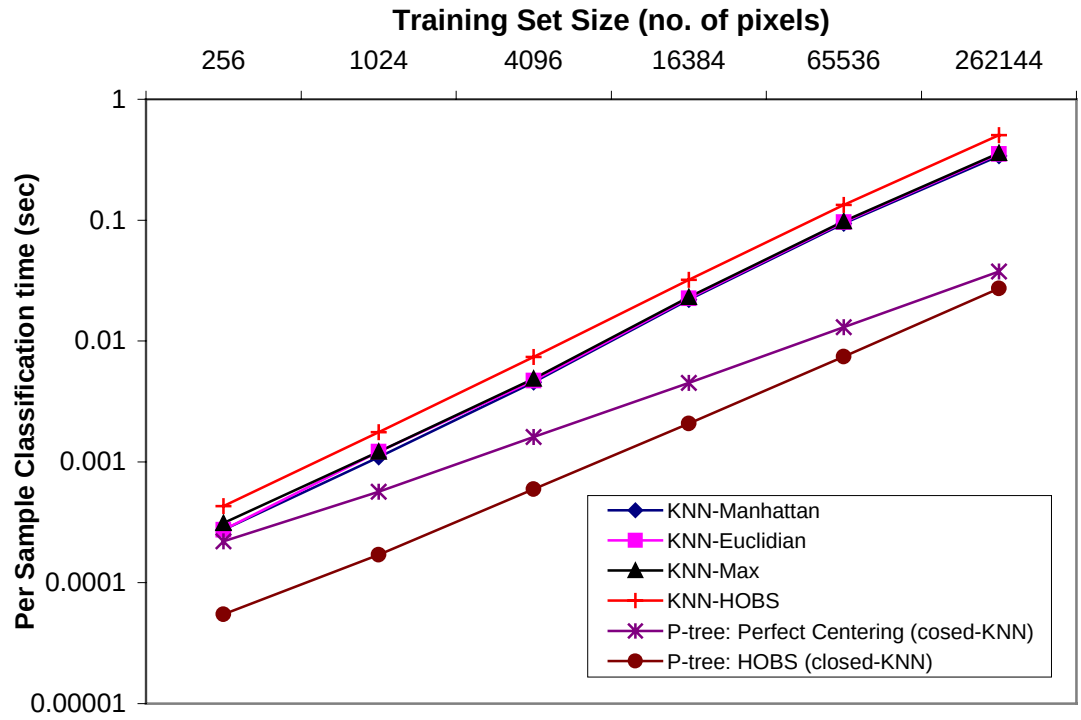
(a) *Max & Euclidian* (b) *Max & Manhattan*

Figure 4.6: *Comparison of neighborhood for different distance metrics (T is the target pixel).*

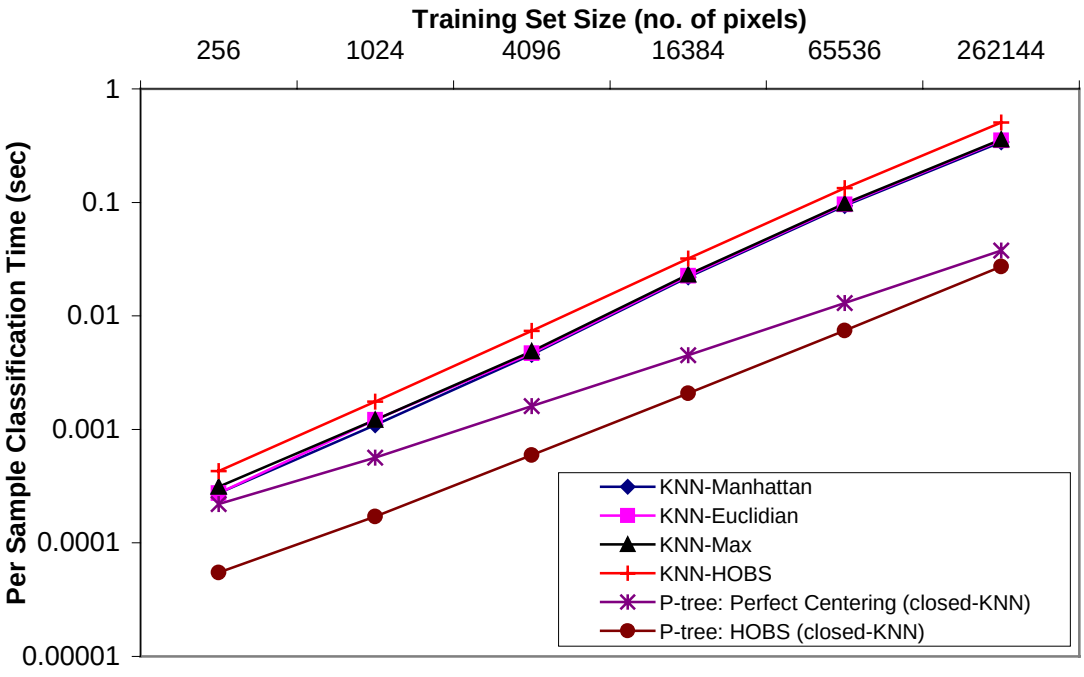
Consider the two points *A* and *B* (figure 4.6). Let, A be a point included in the circle, but not included in the square. Let B be a point, which is included in the square but not the circle. The point A is very similar to target T in the x-dimension but very dissimilar in the y-dimension. On the other hand, the point B is not so dissimilar in any dimension. Relying on high similarity only on one band while keeping high dissimilarity in the other band may decrease the accuracy. Therefore in many cases, inclusion of B in the neighborhood instead of A, is a better choice. That is what we have found for our image data.

We also observe that for almost all of the methods classification accuracy increases with the size of the training dataset. The reason is that with the inclusion of more training pixels, the chance of getting better nearest neighbors increases.

Figure 4.7 shows, on the average, the perfect centering method is five times faster than the KNN and HOBS is 10 times faster (the graphs are plotted in logarithmic scale). Classification times for all of the non-P-tree methods are almost equal. P-tree implementations are more scalable than other methods. Both perfect centering and HOBS increases the classification time with data size at a lower rate than the other



(a) *29NW083097.tiff and associated other files (*1997 *dataset)*



(b) *29NW082598.tiff and associated other files (*1998 *dataset)*

Figure 4.7: *Classification time per sample for the different implementations for the* 1997 *and* 1998 *datasets. Both of the size and classification time are plotted in logarithmic scale.*

methods. For the smaller dataset, the perfect centering method is about 2 times faster than the others and for the larger dataset, it is 10 times faster. This is also true for the HOBS method. The reason is that as dataset size increases, there are more and larger pure-0 and pure-1 quadrants in the P-trees, which increases the efficiency of the ANDing operations.

4.4 Conclusion

In this paper we proposed a new approach to k-nearest neighbor classification for spatial data streams by using a new data structure called the P-tree, which is a lossless compressed and data-mining-ready representation of the original spatial data. Our new approach, called closed-KNN, finds the closure of the KNN set, we call closed-KNN, instead of considering exactly k nearest neighbor. Closed-KNN includes all of the points on the boundary even if the size of the nearest neighbor set becomes larger than k. Instead of examining individual data points to find nearest neighbors, we rely on the expansion of the neighborhood. The P-tree structure facilitates efficient computation of the nearest neighbors. Our methods outperform the traditional implementations of KNN both in terms of accuracy and speed.

We proposed a new distance metric called Higher Order Bit Similarity (HOBS) that provides an easy and efficient way of computing closed-KNN using P-trees while preserving the classification accuracy at a high level.

References

[1] Domingos, P. and Hulten, G., “Mining high-speed data streams”, Proceedings of ACM SIGKDD 2000.

[2] Domingos, P., & Hulten, G., “Catching Up with the Data: Research Issues in Mining Data Streams”, DMKD 2001.

[3] T. Cover and P. Hart, “Nearest Neighbor pattern classification”, IEEE Trans. Information Theory, 13:21-27, 1967.

[4] Dudani, S. (1975). “The distance-weighted k-nearest neighbor rule”. IEEE Transactions on Systems, Man, and Cybernetics, 6:325--327.

[5] Morin, R.L. and D.E.Raeside, “A Reappraisal of Distance-Weighted k-Nearest Neighbor Classification for Pattern Recognition with Missing Data”, IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-11 (3), pp. 241-243, 1981.

[6] J. E. Macleod, A. Luk, and D. M. Titterington. “A re-examination of the distance-weighted k-nearest neighbor classification rule”. IEEE Trans. Syst. Man Cybern., SMC-17(4):689--696, 1987.

[7] Dasarathy, B.V. (1991). “Nearest-Neighbor Classification Techniques”. IEEE Computer Society Press, Los Alomitos, CA.

[8] William Perrizo, "Peano Count Tree Technology", Technical Report NDSU-CSOR-TR-01-1, 2001.

[9] Jiawei Han, Micheline Kamber, “Data Mining: Concepts and Techniques”, Morgan Kaufmann, 2001.

[10] Globig, C., & Wess, S. “Symbolic Learning and Nearest-Neighbor Classification”, 1994.

[11] M. James, “Classification Algorithms”, New York: John Wiley & Sons, 1985.

[12] William Perrizo, Qin Ding, Qiang Ding and Amalendu Roy, “On Mining Satellite and Other Remotely Sensed Images”, Data Mining and Knowledge Discovery 2001, pp. 33-40.

[13] William Perrizo, Qin Ding, Qiang Ding and Amalendu Roy, “Deriving High Confidence Rules from Spatial Data using Peano Count Trees", Proceedings of the 2nd International Conference on Web-Age Information Management, 2001.

CHAPTER 5: PAPER 2

FAST *K*-CLUSTERING ALGORITHM ON SPATIAL DATA USING P-TREES

Abstract

k-clustering is the method of grouping objects into *k* groups with the objective of reducing the intra-cluster squared distance or variance; that is the objects in the same group are similar. We apply *k*-clustering methods to image data with millions of pixels and multi-bands per pixel of data. So far *k*-means is the best algorithm to minimize the total intra-cluster variance. *k*-means is an iterative algorithm that initially selects *k* cluster centers randomly or by some predefined method and, in each iteration, it assigns each pixel to its nearest center and updates the cluster centers using the mean of the clusters. The process of reforming the clusters and their centers requires extensive computation in each iteration, since every data point is typically examined, a distance calculated and a sum formed. To solve the speed issue, in the last decade, various *k*-cluster algorithms such as median-cut, mean-split and variance-based method have been proposed. Median-cut and mean split are fast enough but are not good optimizers. As the variance-based method produces optimization very close to *k*-means but still suffers from speed issue. We propose a new fast P-tree based *k*-clustering method that provides optimization as good as variance based method. Our experiments show that the P-tree implementation of the *k*-clustering algorithm is on the order of 10 times faster than existing implementations. This speed is particularly important in mining data streams.

5.1 Introduction

Clustering is a fundamental problem that arises in many applications in different fields such as data mining, image processing, and bioinformatics [4]. The process of grouping a set of physical or abstract objects into classes of similar objects is called clustering. A cluster is a collection of data objects that are similar to one another within the same cluster and are dissimilar to the objects in other clusters. A cluster of data objects can be treated collectively as one group in many applications [3].

Multi-band image data are increasingly available from variety of sources, including commercial and government satellites, as well as airborne and ground based sensors [1]. This is accompanied by an increased spatial resolution as well as an increased number of bands. A typical image can have millions of pixels with tens of bands per pixel. There are different types of RSI images, such as TM, SPOT, AVHRR, TIFF, etc. For example, a TM image (Thematic Mapper) contains 7 bands, which are B (Blue), G (Green), R (Red), RIR (Reflective-Infrared), MIR (Mid-Infrared), TIR (Thermal Infrared), and MIR2 (Mid_Infrared2) [2]. Typically, a TM scene contains 40M pixels. A TIFF image of agricultural data may contain the Bands red, green, blue, yield, soil moisture, nitrogen, etc. We apply k-means clustering to TIFF images containing 1320×1320 pixels with three 8-bit bands, red, green and blue. That is, more than 5MB of data. The reflectance value in each band ranges from 0 to 255.

The image analyst's challenge is to identify the important and useful features in the image without being overwhelmed by the sheer volume of data. One response to this challenge is provided by algorithms, which segments the image by clustering pixels into classes based on the band similarity of each pixel to other member of the class. As well as

providing the analyst with a picture summarizing the spatial organization of the different band types, these clustering algorithms also provide a very real compression of data [1].

Wan et al [5] described clustering as a procedure that can be viewed as one of finding groupings in a set of events by extremizing some criterion function. In a variety of problems, such unsupervised learning, multivariate data analysis, and digital image processing, the collection of data can be represented as a set of points in a multidimensional vector space. One of the most widely used criterion functions for clustering analysis is the variance or the sum of squared Euclidean distances measured from the cluster centers. The main task in clustering analysis is then to seek the groupings that minimize the sum-of-squared-errors.

$$\text{Sum of the squared error or total variance} = \sum_{i=0}^{k} \sum_{p \in C_i} d_2^2(c_i, p)$$

Where k is the number of clusters, c_i is the center or mean of the cluster C_i and $d_2^2(c_i, p)$ is the squared Euclidian distance of point p from the cluster center or mean c_i.

In [4], Yair Bartal et al provided an algorithm to minimize the sum of the pair-wise squared distance of the points in the clusters.

$$\text{Sum of the pair-wise squared distance} = \sum_{i=0}^{k} \sum_{p,q \in C_i} d_2^2(p, q)$$

In this paper we provide an algorithm to minimize the sum of the squared error or total variance. The problem of finding the global minimum solution is NP-complete [6]. To solve the problem, a number of approximate clustering algorithms developed [4, 7, 8, 9, 10]. These techniques can be divided into two categories: iterative optimization and divisive approach. K-means is a iterative optimization procedure that is most frequently

used for finding a local minimum solution. The converging time required by the k-means approaches can easily become unmanageable, particularly for a large clustering problem. However k-means provides the best optimization among the algorithms mentioned.

On the other hand, a heuristic approach based divisive techniques tries to reduce the computational complexity and at the same time produce and acceptable solution. Divisive approaches initially assume there is a one cluster containing all of the given data points. Then divide it into two clusters and recursively it selects one of the clusters chosen by some heuristic method and divide it into another two clusters; continues this until process until there are k clusters. Median-cut [7], mean-split [9], variance-based k-Clustering [4] are important among divisive algorithms. In section 5.2 we reviewed this methods. All of these divisive approaches partition a cluster by a hyperplane perpendicular to one of the coordinate axes. Variance-based method gives a better approximation to k-means but slower than the other two methods. This method finds the optimal cut-point in an axis by examining each point in the axis. Instead we propose a method to find the optimal cut point by k-means type convergence techniques that avoids examination of each point in the axis. This makes our algorithm faster than variance-based method while keeping the cluster quality as good as variance-based method. Our method can be called a hybrid method that uses the k-means type convergence technique in the divisive approach. Our main improvement in speed is using P-trees to compute the sum and variance in each step of the algorithm without scanning databases. This makes our algorithm order of 10 times faster than the other methods. Details of our algorithm are discussed in section 5.3.

5.2 Review of the k-Clustering Algorithms

5.2.1 k-Means Algorithm

The k-means iterative procedure has received considerable attention in clustering analysis since it produces a very good minimization of the sum-of-the-squared-error or total variance function. This algorithm first selects *k* initial cluster centers. Then, the *k* clusters are formed by associating each data point with its closest cluster center. The centroids or means of these K clusters become the new cluster centers. The above procedure is repeated until the new cluster centers are the same as the previous ones. Although the k-means algorithm has been widely used in many applications, it has been shown only recently [11] that it converges to a local minimum solution in a finite number of iterations if a quadratic metric is used.

With the spatial-storage scheme, the time complexity of this algorithm is proportional to $O(mTNK)$. The number of iterations T necessary for the algorithm to converge depends on the distribution of the data points, the number of clusters required, the size of the space, and the choice of initial cluster centers. For a large clustering problem, the computation can be very costly. For example, it may take more than twenty hours on a VAX 780 computer to reduce 256 clusters for a full-color image [9].

5.2.2 The Mean-Split Algorithm

Considering the whole data space, a big hyperbox, as the initial cluster, all of the divisive techniques split the hyperboxes with a hyperplane perpendicular to an axis until there are *k* hyperboxes. All of the techniques share the following common steps.

a) Based on some heuristic select a hyperbox to split.

b) Select an axis along which the hyperbox to be split.

c) Select the cut point on the selected axis.

d) Split the hyperbox into two sub-hyperboxes using a hyperplane perpendicular to the selected hyperbox at the selected cut-point.

e) Repeat the process until there are k hyper boxes. Each hyperbox is a cluster.

The different divisive approaches differ in different strategies in selecting the hyperbox, axis and cut-point.

In mean-split algorithm, when a hyperbox is created by splitting, it is assigned a number of clusters. If the number of clusters assigned to cluster is more than one, it is selected to split. The initial big hyperbox is assigned all of the k clusters. If L be the number of clusters assigned to a hyperbox, when the hyperbox is split into two sub-hyperboxes, then L_i clusters are assigned to the i^{th} sub-hyperbox according to the following formula:

$$L_i = L\left[\alpha\frac{n_i}{n_1+n_2}+(1-\alpha)\frac{V_1}{V_1+V_2}\right], \qquad i = 1, 2$$

where n_i is the number of data points in sub-hyperbox i and V_i is the volume of the hyperbox i. The parameter α is restricted to the range $0.5 \le \alpha \le 0.7$.

Min-split algorithm selects the axis that has the projected distribution of the data points with the largest spread and the mean of the projected distribution is the cut-point.

The main advantage of the mean-split algorithm is that it has a lower computational cost: $O(mN \log k)$ for time and $O(mN)$ for space; m is the dimensionality of the data and N is the total number of data points. But there are draw backs associated with this algorithm. To partition a hyperbox by a plane passing through the mean does not necessarily produce a lower quantization error.

5.2.3 Variance-Based Algorithm

Variance-based algorithm is another divisive algorithm that selects the hyperbox with the largest variance to split. Then it takes the projection of the data points along each dimension and examines each point in that dimension to find which cut-point gives the largest variance reduction in the projected data. That cut-point and axis are chosen to split the hyperbox.

Letting μ and σ^2 be the mean and variance or the projected 1-dimensional distribution, the optimal cut-point t_{opt} is defined by:

$$t_{opt} = \arg\max_t \left[\sigma^2 - w_1\sigma_1^2(t) - w_2\sigma_2^2(t)\right]$$

where w_i and $\sigma_i^2(t)$ are the weight and variance of the i^{th} interval (i = 1, 2). t_{opt} can also be expressed as $t_{opt} = \arg\max_t \left[\frac{w_1}{w_2}\{\mu - \mu_1(t)\}\right]$.

This algorithm produces better minimization of the quantization error, which is closer to that of k-means algorithm but suffers from the higher computational cost, which is $O(mNk)$, since almost every point along an axis is examined.

5.3 Our Algorithm

We propose an approximate divisive algorithm that arbitrarily chooses a cut-point and iteratively converges to the optimal cut-point instead examining every single point in the interval and we compute the mean and the variance by using P-trees with scanning the databases that makes it faster than any algorithm discussed above. In terms of quality of the clusters, our algorithm produces as good minimization as variance-based method.

Finding the optimal cut-point on an axis includes the following steps:

a) take the projection of the data point on the specified axis that forms 1-dimensional data.

b) Arbitrarily select two points m_1 and m_2 ($m_1 < m_2$) as initial means on the axis.

c) Assign data point p to the cluster C_1 if $p \leq (m_1 + m_2) / 2$, otherwise assign p to C_2.

d) Update m_1 and m_2 with the means of C_1 and C_2.

e) Repeat the process until there is no change in m_1 and m_2.

f) The optimal cut-point is $(m_1 + m_2) / 2$.

Select the hyperbox and the axis that gives the maximum difference between m_1 and m_2 i.e. max $(m_2 - m_1)$ and split the hyperbox through the cut point $(m_1 + m_2) / 2$. At each splitting we do not need to compute the optimal cut-point for each of hyperboxes. At the time of forming the two new hyperboxes by splitting some big hyperbox, we find its optimal cut-point and axis and store it until these hyperboxes are split further.

The number of iterations to find the cut-point is much less than the number of points along the axis and the computation required in each iteration is as same as the computation required for each examining point in the variance-based method. Therefore our method finds the optimal cut-point in fewer steps than the variance based method.

We represent each cluster by a template P-tree. A **template P-tree** represents a subset of the data points or pixels. The template P-tree, P_t, contains a 1 bit for a pixel if the pixel is in that subset, otherwise there is a 0 bit in P_t for that pixel. In our algorithm, the P-tree, P_{ci}, representing the cluster C_i is the template P-tree for C_i. The template P-tree for initial cluster, P_{C1}, is a pure1-tree (defined in section 3.2) i.e. C_1 contains all of the pixels.

In step (c) of the above algorithm we do not compare each individual data point p to assign to a cluster. Suppose we want to find optimal cut-point along the axis j of the cluster C_i. We compute the interval P-tree $P_j(0, (m_1+m_2)/2)$ and $P_j((m_1+m_2)/2, \text{UPPER})$, where UPPER is largest possible band-value. Then the two new clusters are:

$P_{Ci1} = P_{Ci}$ & $P_j(0, (m_1+m_2)/2)$

$P_{Ci2} = P_{Ci}$ & $P_j((m_1+m_2)/2, \text{UPPER})$

And note that $P_j((m_1+m_2)/2, \text{UPPER}) = P'_j(0, (m_1+m_2)/2)$

To compute interval P-trees by ORing value P-trees for each of the values in the interval is costly if the length of the interval is large. We provide an optimal algorithm (Algorithm 5.1) to compute interval P-tree by decomposing the interval into few subintervals such that each of the sub-intervals can be computed in less cost than the computation of a value P-tree.

Algorithm 5.1 Computing Interval P-trees

Input: An interval $[v_1, v_2]$ of the band i and the basic P-trees for the band i, $P_{i,j}$, for j = 0, 1, 2, …, b – 1, where b is the number of bits in band i.

Output: P_{int}, the interval P-tree $P_i(v_1, v_2)$.

1. Initialize P_{int} by pure0 tree P^0: $P_{int} \leftarrow P^0$
2. Find an integer $n \in [v_1, v_2+1]$, such that n is divisible by 2^t for some non-negative integer t and no integer in the interval $[v_1, v_2+1]$ is divisible by 2^{t+1}. Without examining each number in the interval, n can be calculated by the following steps.

```
n ← 1
while 2n ≤ v2 + 1  do
        n ← 2n
m ← n / 2
while n < v1 do
```

while $n + m > v_2 + 1$ *do*

$m \leftarrow m / 2$

$n \leftarrow n + m$

3. a) Initialize n_1 by n: $n_1 \leftarrow n$

   b) Find a positive integer r such that $n_1 - 2^r \geq v_1$ and $n_1 - 2^{r+1} < v_1$

   c) $P_{sub} \leftarrow P_{i,0} \,\&\, P_{i,1} \,\&\, P_{i,2} \,\&\, \ldots \&\, P_{i,b\text{-}r\text{-}1}$, P_{sub} is the interval P-tree for the sub-interval $[n_1 - 2^r, n_1 - 1]$, i.e. $P_i(n_1 - 2^r, n_1 - 1)$.

   $P_{int} \leftarrow P_{int} \,\&\, P_{sub}$

   $n_1 \leftarrow n_1 - 2^r$

   d) If $n_1 > v_1$, repeat step b and c.

4. a) Initialize n_2 by n: $n_2 \leftarrow n$

   b) Find a positive integer r such that $n_2 + 2^r \leq v_2 + 1$ and $n_1 + 2^{r+1} > v_2 + 1$

   c) $P_{sub} \leftarrow P_{i,0} \,\&\, P_{i,1} \,\&\, P_{i,2} \,\&\, \ldots \&\, P_{i,b\text{-}r\text{-}1}$, P_{sub} is the interval P-tree for the sub-interval $[n_2, n_2 + 2^r - 1]$, i.e. $P_i(n_2, n_2 + 2^r - 1)$.

   $P_{int} \leftarrow P_{int} \,\&\, P_{sub}$

   $n_2 \leftarrow n_2 + 2^r$

   d) If $n_2 \leq v_2$, repeat step b and c.

To compute the mean and variance of a cluster or interval we perform AND operations on P-trees and use the root counts of the P-trees instead of scanning the databases. ANDing of P-trees is very much faster than the database scanning.

5.3.1 Computation of Sum and Mean from the P-trees

If the basic P-trees that are constructed from the original dataset are given, we compute the sum of the band-values of the pixels by $\sum_{j=0}^{n-1} 2^{n-j-1} rc(P_{i,j})$ (the notations of the symbols have been given in section 3.2 of chapter 3). If the number of bits, *n*, in each band-value is 8 (n=8), then it requires the computation of only 8 terms. The root of P-trees, $rc(P_{i,j})$ can directly be found in the header of a P-tree (section 3.3). The proof of the expression is given in theorem 5.1. The mean can be calculated by dividing sum by the number of pixels, *N*. Theorem 5.1 calculates the sum of band-values for all of the pixels. But after dividing the first, cluster, which contains all of the pixels in the space, to calculate the sum for the pixels that are included in a particular cluster, we use

$$\text{sum} = \sum_{j=0}^{n-1} 2^{n-j-1} rc(P_{i,j} \,\&\, P_t) \quad \text{(theorem 5.2)}$$

$$\text{mean} = \frac{\sum_{j=0}^{n-1} 2^{n-j-1} rc(P_{i,j} \,\&\, P_t)}{rc(P_t)}$$

where P_t is the template **P-tree** or **mask P-tree**.

Lemma 5.1: $\sum_{x=0}^{c-1}\sum_{y=0}^{r-1} b_{x,y,i,j} = rc(P_{i,j})$.

Proof: $\sum_{x=0}^{c-1}\sum_{y=0}^{r-1} b_{x,y,i,j}$

= the number of pixels having '1' in the j^{th} bit of i^{th} band.

= the number of 1s in the P-tree $P_{i,j}$

= $rc(P_{i,j})$.

Theorem 5.1: *Summation of the values for bands i of all pixels,* $S_i = \sum_{j=0}^{n-1} 2^{n-j-1} rc(P_{i,j})$, *where n, c and r are the number of bits in band i, columns and rows respectively.*

Proof: $V_{x,y,i} = b_{x,y,i,0}2^{n-1} + b_{x,y,i,1}2^{n-2} + b_{x,y,i,2}2^{n-3} + ... + b_{x,y,i,n-1}2^0$

$$= \sum_{j=0}^{n-1} b_{x,y,i,j} 2^{n-j-1}$$

and $S_i = \sum_{x=0}^{c-1}\sum_{y=0}^{r-1} V_{x,y,i}$

$$= \sum_{x=0}^{c-1}\sum_{y=0}^{r-1}\left(\sum_{j=0}^{n-1} b_{x,y,i,j} 2^{n-j-1}\right)$$

$$= \sum_{j=0}^{n-1}\left(\sum_{x=0}^{c-1}\sum_{y=0}^{r-1} b_{x,y,i,j} 2^{n-j-1}\right), \text{ since addition is associative}$$

$$= \sum_{j=0}^{n-1} 2^{n-j-1}\left(\sum_{x=0}^{c-1}\sum_{y=0}^{r-1} b_{x,y,i,j}\right), \text{ since } 2^{n-j-1} \text{ is independent of } x \text{ and } y.$$

Therefore $S_i = \sum_{j=0}^{n-1} 2^{n-j-1} rc(P_{i,j})$. $\langle \text{Lemma 5.1} \rangle$

Lemma 5.2: *For single-bit operands then bit-wise 'AND' operation (&) and multiplication produce the same result and the result is also a* 1*-bit value; that is* $a \times b = a \,\&\, b$ *if a and b are 1-bit values i.e. either* 1 *or* 0.

Proof: Using a truth-table to examine all possible combinations of values of *a* and *b*

a	b	$a \times b$	$a \,\&\, b$
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

The data from the table show that for any 1-bit values *a* and *b*, $a \times b = a \,\&\, b$. And both ($a \times b$) and ($a \,\&\, b$) are 1-bit value.

Theorem 5.2: *Summation of the values for band i of the pixels represented by the* ***template P-tree*** *or* ***mask P-tree****,* P_t*, is* $S_i(P_t) = \sum_{j=0}^{n-1} 2^{n-j-1} rc(P_{i,j} \,\&\, P_t)$.

Proof: Let $t_{x,y}$ is the bit stored in P_t for the pixel $p_{x,y}$

i.e. $t_{x,y}$ = 1, if $p_{x,y}$ is in the group of pixels represented by P_t

= 0, otherwise

Then, $S_i(P_t) = \sum_{x=0}^{c-1}\sum_{y=0}^{r-1} V_{x,y,i}\, t_{x,y}$

$$= \sum_{x=0}^{c-1}\sum_{y=0}^{r-1}\left(\sum_{j=0}^{n-1} b_{x,y,i,j}\, 2^{n-j-1}\right) t_{x,y}$$

$$= \sum_{x=0}^{c-1}\sum_{y=0}^{r-1}\left(\sum_{j=0}^{n-1} b_{x,y,i,j}\, 2^{n-j-1}\, t_{x,y}\right)$$

$$= \sum_{j=0}^{n-1}\left(\sum_{x=0}^{c-1}\sum_{y=0}^{r-1} b_{x,y,i,j}\, 2^{n-j-1}\, t_{x,y}\right)$$

$$= \sum_{j=0}^{n-1} 2^{n-j-1}\left(\sum_{x=0}^{c-1}\sum_{y=0}^{r-1} b_{x,y,i,j}\, t_{x,y}\right)$$

$$= \sum_{j=0}^{n-1} 2^{n-j-1}\left(\sum_{x=0}^{c-1}\sum_{y=0}^{r-1} b_{x,y,i,j} \;\&\; t_{x,y}\right) \qquad \langle \text{Lemma 5.2} \rangle$$

$$= \sum_{j=0}^{n-1} 2^{n-j-1} rc(P_{i,j} \,\&\, P_t) \qquad \langle \text{Lemma 5.1} \rangle$$

5.3.2 Computation of Variance from the P-trees

Variance is defined by $\frac{1}{N}\sum(x-\mu)^2$ that can be simplified to $\frac{1}{N}\sum x^2-\mu^2$. Theorem 5.1 and 5.2 allow us to compute mean, μ, from the P-tree; now if we can find $\sum x^2$ using the P-tree, then we do not need to scan databases to perform any computation required by the algorithm. Theorem 5.3 and 5.4 facilitates the computation of the sum of the squared band-values.

Lemma 5.3: $\sum_{x=0}^{n-1}\sum_{y=0}^{x-1} f(x,y)=\sum_{x=0}^{n-1}\sum_{y=x+1}^{n-1} f(y,x)$

Proof: $\sum_{x=0}^{n-1}\sum_{y=0}^{x-1} f(x,y)=\sum_{y=0}^{0} f(1,y)+\sum_{y=0}^{1} f(2,y)+\sum_{y=0}^{2} f(3,y)+\cdots+\sum_{y=0}^{n-2} f(n-1,y)$

$= f(1,0)$

$+ f(2,0)+ f(2,1)$

$+ f(3,0)+ f(3,1)+ f(3,2)$

…

$+ f(n-1,0)+ f(n-1,1)+ f(n-1,2)+\cdots+ f(n-1,n-2)$

$= \sum_{y=1}^{n-1} f(y,0)+\sum_{y=2}^{n-1} f(y,1)+\sum_{y=3}^{n-1} f(y,2)+\cdots+\sum_{y=n-1}^{n-1} f(y,n-2)$ <Accumulating terms over a vertical line>

$=\sum_{x=0}^{n-1}\sum_{y=x+1}^{n-1} f(y,x)-\sum_{y=n}^{n-1} f(y,x)$

$=\sum_{x=0}^{n-1}\sum_{y=x+1}^{n-1} f(y,x)$

Theorem 5.3: *Summation of the squared values for bands i of all pixels,* $S(v_i^2) = \sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 2^{2n-j-k-2}\, rc(P_{i,j} \,\&\, P_{i,k})$, *where n, c and r are the number of bits in band i, columns and rows respectively.*

Proof: $V_{x,y,i} = \sum_{j=0}^{n-1} b_{x,y,i,j}\, 2^{n-j-1}$

$$\Rightarrow V_{x,y,i}^2 = \left(\sum_{j=0}^{n-1} b_{x,y,i,j}\, 2^{n-j-1}\right)\left(\sum_{j=0}^{n-1} b_{x,y,i,j}\, 2^{n-j-1}\right)$$

$$= \left(\sum_{k=0}^{n-1} b_{x,y,i,k}\, 2^{n-k-1}\right)\left(\sum_{j=0}^{n-1} b_{x,y,i,j}\, 2^{n-j-1}\right), \quad \text{replacing the index } j \text{ by } k \text{ in the left summation}$$

$$= \sum_{j=0}^{n-1}\left(\sum_{k=0}^{n-1} b_{x,y,i,k}\, 2^{n-k-1}\right) b_{x,y,i,j}\, 2^{n-j-1}$$

$$= \sum_{j=0}^{n-1}\left\{\sum_{k=0}^{n-1}\left(b_{x,y,i,k}\, 2^{n-k-1}\right)\left(b_{x,y,i,j}\, 2^{n-j-1}\right)\right\}$$

$$= \sum_{j=0}^{n-1}\sum_{k=0}^{n-1} b_{x,y,i,k} b_{x,y,i,j}\, 2^{2n-j-k-2}$$

Now $S(v_i^2) = \sum_{x=0}^{c-1}\sum_{y=0}^{r-1} V_{x,y,i}^2$

$$= \sum_{x=0}^{c-1}\sum_{y=0}^{r-1} b_{x,y,i,k} b_{x,y,i,j}\, 2^{2n-j-k-2}$$

$$= \sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 2^{2n-j-k-2}\left(\sum_{x=0}^{c-1}\sum_{y=0}^{r-1} b_{x,y,i,k} b_{x,y,i,j}\right)$$

$$= \sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 2^{2n-j-k-2}\left(\sum_{x=0}^{c-1}\sum_{y=0}^{r-1} b_{x,y,i,k} \,\&\, b_{x,y,i,j}\right) \qquad \langle \text{Lemma } 5.2 \rangle$$

$$= \sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 2^{2n-j-k-2}\, rc\left(P_{i,j}\ \&\ P_{i,k}\right) \qquad \langle \text{Lemma } 5.1 \rangle$$

Note: Using theorem 5.3, to calculate $S\left(v_i^2\right)$, the required number of '&' operations is n^2, which can be reduced to ½ ($n^2 - n$) by using the corollary 5.1.

Corollary 5.1: $S\left(v_i^2\right)$ can be expressed as $\sum_{j=0}^{n-1}\left(\sum_{k=0}^{j-1} 2^{2n-j-k-1}\, rc\left(P_{i,j}\ \&\ P_{i,k}\right) + 2^{2(n-j-1)}\, rc\left(P_{i,j}\right)\right)$.

Proof: $S\left(v_i^2\right) = \sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 2^{2n-j-k-2}\, rc\left(P_{i,j}\ \&\ P_{i,k}\right)$

$$= \sum_{j=0}^{n-1}\left(\sum_{k=0}^{j-1} 2^{2n-j-k-2}\, rc\left(P_{i,j}\ \&\ P_{i,k}\right) + \sum_{k=j}^{j} 2^{2n-j-k-2}\, rc\left(P_{i,j}\ \&\ P_{i,k}\right) + \sum_{k=j+1}^{n-1} 2^{2n-j-k-2}\, rc\left(P_{i,j}\ \&\ P_{i,k}\right)\right)$$

$$= \sum_{j=0}^{n-1}\left(\sum_{k=0}^{j-1} 2^{2n-j-k-2}\, rc\left(P_{i,j}\ \&\ P_{i,k}\right) + 2^{2(n-j-1)}\, rc\left(P_{i,j}\ \&\ P_{i,j}\right) + \sum_{k=j+1}^{n-1} 2^{2n-k-j-2}\, rc\left(P_{i,j}\ \&\ P_{i,k}\right)\right)$$

$\langle$ & is idempotent and commutative i.e. $P_{i,j}\ \&\ P_{i,j} = P_{i,j}$ and $P_{i,k}\ \&\ P_{i,j} = P_{i,j}\ \&\ P_{i,k}$ $\rangle$

$$= \sum_{j=0}^{n-1}\left(\sum_{k=0}^{j-1} 2^{2n-j-k-2}\, rc\left(P_{i,j}\ \&\ P_{i,k}\right) + 2^{2(n-j-1)}\, rc\left(P_{i,j}\right) + \sum_{k=j+1}^{n-1} 2^{2n-k-j-2}\, rc\left(P_{i,k}\ \&\ P_{i,j}\right)\right)$$

$$= \sum_{j=0}^{n-1}\sum_{k=0}^{j-1} 2^{2n-j-k-2}\, rc\left(P_{i,j}\ \&\ P_{i,k}\right) + \sum_{j=0}^{n-1} 2^{2(n-j-1)}\, rc\left(P_{i,j}\right) + \sum_{j=0}^{n-1}\sum_{k=j+1}^{n-1} 2^{2n-k-j-2}\, rc\left(P_{i,k}\ \&\ P_{i,j}\right)$$

$$= \sum_{j=0}^{n-1}\sum_{k=0}^{j-1} 2^{2n-j-k-2}\, rc\left(P_{i,j}\ \&\ P_{i,k}\right) + \sum_{j=0}^{n-1} 2^{2(n-j-1)}\, rc\left(P_{i,j}\right) + \sum_{j=0}^{n-1}\sum_{k=0}^{j-1} 2^{2n-k-j-2}\, rc\left(P_{i,j}\ \&\ P_{i,k}\right) \qquad \langle \text{Lemma } 5.3 \rangle$$

$$= \sum_{j=0}^{n-1}\left(2\sum_{k=0}^{j-1} 2^{2n-j-k-2}\, rc\left(P_{i,j}\ \&\ P_{i,k}\right) + 2^{2(n-j-1)}\, rc\left(P_{i,j}\right)\right)$$

$$= \sum_{j=0}^{n-1}\left(\sum_{k=0}^{j-1} 2^{2n-j-k-1}\, rc\left(P_{i,j}\ \&\ P_{i,k}\right) + 2^{2(n-j-1)}\, rc\left(P_{i,j}\right)\right)$$

Theorem 5.4: *Summation of the squared values for band i of the pixels represented by the* ***template P-tree*** *or* ***mask P-tree****,* P_t*, is* $S(v_i^2, P_t) = \sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 2^{2n-j-k-2} rc(P_{i,j} \,\&\, P_{i,k} \,\&\, P_t)$.

Proof: Proof can easily be constructed by following the techniques used in proving the theorems 5.2 and 5.3.

5.4 Conclusion

In this paper we proposed a new k-Clustering algorithm that combined the convergence techniques of k-means algorithm into the divisive k-clustering techniques using P-trees to produce a fast clustering techniques that minimze the sum of the squared error. Our new algorithm is faster than any k-clustering algorithms while the optimization of the sum of the squared error is as good as the variance-based method.

We provided theorems to calculate the mean and variance from the P-trees without scanning the database that saves computational time significantly. In this paper we also included an optimal algorithm to compute the interval P-trees.

References

[1] Mike Estlick, Mirian Leeser, James Theiler and John Szymanski, “Algorithmic Transformations in the Implementation of K-means Clustering on Reconfigurable Hardware,” FPGA 2001, February 11-13, 2001, Monterey, CA, USA.

[2] William Perrizo, Qin Ding, Qiang Ding and Amalendu Roy, "On Mining Satellite and Other Remotely Sensed Images", ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'01), May 2001, pp. 33-40.

[3] Jiawei Han and Micheline Kamber, “Data Mining: Concepts and Techniques,” Morgan Kaufmann Publishers, August 2000. ISBN 1-55860-489-8

[4] Yair Bartal, Moses Charikary, Danny Razz, “Approximating min-sum k-Clustering in Metric Spaces,” Symposium on Theory of computing, July 6-8, 2001, Hersonissos, Crete, Greece.

[5] S. J. WAN, S. K. M. WONG, and P. PRUSINKIEWICZ, “An Algorithm for Multidimensional Data Clustering.”

[6] L. Hyafil and R. L. Rivest, “Construction Optimal Binary Decision Tree is NP-Complete.” Inf. Process. Lett. May 1976, 15-17

[7] P. Heckbert, “Color Image Quantization for Frame Buffer Display .” ACM Trans. Comput. Gr. 16, 3 (July 1982), 297-307

[8] J. B. MacQueen, “Some Methods for Classification and Analysis of Multivariate Observations.” Proceedings of 5th Berkley Symposium on Mathematical Statistics and Probability 1 (1967), 281-297

[9] X. Wu and I. H. Witten, “A Fast k-Means Type Clustering Algorithm.” Dept of Computer Science, Univ. of Calgary, Canada, May 1985

[10] M. Inaba, N. Katoh and H. Imao, “Application of Weighted Voronoi Diagrams and Randomization to Variance-Based k-Clustering.” 10th ACM Computational Geometry 94-6/94 Stony Brook, NY, USA.

[11] S. Z. Selim and M. A. Ismail. “k-Means-Type Algorithms: A Generalized convergence theorem and characterization of the local optimality.” IEEE transaction, Pattern Analysis Machine Intelligence.” PAMI-6, 1, 1994, 81-87.

CHPATER 6: GENERAL CONCLUSION

In this thesis we analyzed distance metric-based data mining techniques using P-trees. Various distance metrics are considered and their behaviors are analyzed. We developed the fast techniques of computing neighborhoods and partitions, which made by the decision boundaries of the distance metrics, using P-trees. We successfully used those techniques to develop fast classification and clustering algorithms.

A new distance metric is proposed that can be used efficiently in P-tree-based computation. We included the proof that the new proposed distance metric satisfies the criteria of the distance metrics.

Some interesting and useful properties of P-trees have been revealed. We found proved some useful theorems that enable statistical computations such as finding sum, mean and variance from the P-trees without scanning databases.

BIBLIOGRAPHY

A. William Perrizo, "Peano Count Tree Technology Lab Notes", Computer Science Department, North Dakota State University, Fargo, ND, USA.

B. Jiawei Han and Micheline Kamber, "Data Mining: Concepts and Techniques," Morgan Kaufmann Publishers, August 2000. ISBN 1-55860-489-8

C. "Conic Sections in Taxicab Geometry," available from http://user3.stritch.edu/~ber/Taxi/taxi.html.

D. "Sample difference metrics used in ANALOG," available from http://geochange.er.usgs.gov/pub/tools/analog/doc/distance.html.

E. "Unsupervised Pattern Classification," available from http://soft.ee.umist.ac.uk/nigel/neural/SOM5.html.

F. "Welcome to the GeneX^{tm} Clustering Analysis Page," available from http://genex.ncgr.org/genex/rcluster/help.html.

G. "Distance Metrics", available from http://www.dai.ed.ac.uk/HIPR2/metric.htm.

H. "Geometry on a Smooth Manifold," available from http://www.chaos.org.uk/~eddy/math/smooth/geometry.html.