Dissertation Proposal: Exploration of the High Dimensional Data

Jian Lu
Computer Science Department
University of Massachusetts Lowell
Sept. 10th, 2013
OUTLINE

• Introduction
• Background
• Problem Definition
• Our Approach
• Future Work
• Conclusion
Introduction

This research study focuses on the organization and analysis of the high dimensional data.

The term ‘big data’ has been becoming more and more popular because it brings us both challenges and benefits. We consider ‘BIG’ in two ways: the large size of datasets and complex features of the data objects.

In my research, we consider the latter as data of the high dimensionality.
OUTLINE

• Introduction
• Background
  – Data Everywhere
  – Knowledge Discovery
  – Motivation: High Dimensional Trend
• Problem Definition
• Our Approach
• Future Work
• Conclusion
Background: Data Everywhere

• **The century of data**
  - The Digital Universe, 40 trillion GB, double every 2 years
  - Industrial revolution of data
    - NoSQL database
    - Hadoop + Parallel Processing

• **Huge benefits, but also big headache.**
  - Business trends, prevent diseases, combat crimes, etc...
  - Hard to work with
    - Superabundant: Wal-Mart transactions, 2500 TB per hour
    - Complex: Gene expression, tens of thousands of features
Knowledge Discovery

“Information is not knowledge.”

-- Albert Einstein

Data Acquire
- Data Selection
- Data Cleaning
- Data integration

Organization
- Storage System
- Indexing Method

Data Mining
- Analyzing
- Modeling
- Evaluation

Knowledge
- Interpretation
- Visualization
- Making Decision
Motivation: Data Trends

• High Dimensionality
  – Finance & Economics
  – Biology & Medicine
  – Astronomy
  – Multimedia data
  – Moving objects
  – Web term document
  – …
OUTLINE

• Introduction
• Background
• Problem Definition
  – Curse of Dimensionality
  – Examples
  – Research Objectives
• Our Approach
• Future Work
• Conclusion
Curse of Dimensionality

- The “curse of dimensionality” is coined by Bellman in 1961.

- Various phenomenon in several fields
  - Mathematics: it refers to the intractability of accurately approximating a general high dimensional function for given data.
  - Data analysis: it refers to the phenomena that techniques that work well at lower dimensions perform poorly as the dimensionality increases.
  - Database: it refers to the problem of indexing and searching data through a high dimensional space.
  - Sampling, meaningless distance, etc...

- Generally, when the dimensionality increases, the hyper-volume of the space increases so fast that the available data becomes sparse.
Example 1: Multidimensional data

- **R-tree based indexing**
  - Most common used spatial access method
  - “R” means Minimum Bounding Rectangle (MBR)
  - Perform well at lower dimensionalities
  - **Slowdown rapidly with increasing dimensionality:** exponential growth of the coverage and overlap as a function of dimensionality

![Diagram showing R-tree indexing at different dimensions](image)
Example 2: Distance Function

- **Euclidean Distance**
  - Meaningless in high dimensional space
  - Volume ratio of Hypersphere (R=r) to Hypercube (L=2r)
  - All points are “far away” from the center

\[
\frac{\pi^{d/2}}{d 2^{d-1} \Gamma(d/2)}
\]

<table>
<thead>
<tr>
<th>d</th>
<th>1</th>
<th>0.785</th>
<th>0.523</th>
</tr>
</thead>
<tbody>
<tr>
<td>d=1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d=2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d=3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Research Objectives

- **Attack the curse of dimensionality**
- **Organize high dimensional data**
  - Indexing, searching, etc...
- **High dimensional data analysis**
  - Clustering, understanding, etc...
OUTLINE

• Introduction
• Background
• Problem Definition
• Our Approach
  – Cantor Pairing Function
  – PL-Tree Algorithms
  – 2D Example
  – Optimization
  – Performance Evaluation
• Future Work
• Conclusion
Our Approach: PL-Tree

- **Indexing method for accessing high-dimensional data**

- **Similarity searching queries**
  - Point queries: particular feature values
  - Range queries: fall within a given range
  - Nearest neighbor queries: similar to a given query object

- **Curse of dimensionality for similarity searching**
  - Number of objects to be examined grows exponentially with dimension.
Cantor Pairing Function

- **Algebraic technique**
- **Reduce effect of “curse of dimensionality”**
- **Recursive $k$-degree CPF ($f_k : \mathbb{N}^k \rightarrow \mathbb{N}$):**

  \[
  f_k(x_1, \ldots, x_k) = \begin{cases} 
  f_2(x_1, f_{k-1}(x_2, \ldots, x_k)), & \text{if } k > 2 \\
  \frac{1}{2}(x_1 + x_2)(x_1 + x_2 + 1), & \text{if } k = 2 
  \end{cases}
  \]

- **Counting hyperplane** $x_1 + \ldots + x_k = h$
CPF distance

- **2D example** – $f_2(x,y)$

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>6</th>
<th>...</th>
<th>...</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>7</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>y/x</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

- **CPF distance**
PL-Tree

- Index high-dimensional data based on hierarchical CPF distances
  - CPF is bijection: $N^k \rightarrow N$
  - Computational cost: $O(k)$, $k$ is the number of dimensions

- Coordinate system of the $k$-dimensional space
  - Home-system: non-negative, real
  - Rescaling vector: $U = (u_1, \ldots, u_k)$
  - Rescaling function: $S_U(D) = \left(\left\lfloor d_1 / u_1 \right\rfloor, \ldots, \left\lfloor d_k / u_k \right\rfloor\right)$, where $D = (d_1, \ldots, d_k)$
  - Label of $D$ in home system: $L = f_k(S_U(D))$
  - Sub-system & sub-hypercube
PL-Tree: Data Structure

Partition a hypercube into sub-hypercubes

Label each sub-hypercube

Recursive procedure if object # > Thres

Tree-like structure of leaf & directory nodes
PL-Tree Nodes

- **Hypercube** = node on PL-Tree
- **Leaf node**: bounding box + data ID
- **Directory node**: \((B, U, E)\)
  - \(B\) is the bounding box
  - \(U\) is a rescaling vector under the home system of the node (hypercube)
  - \(E\) is a list of child entries of form \((L, ptr)\) sorted by the label value \(L\) (CPF distance)
PL-Tree: 4-Step Process

- **PL = Partitioning-and-Labeling process**
  - Partition the hypercube (home system) by $U$
  - Label data in the same sub-hypercube (sub system) with the same CPF distance

**Partitioning**
- Objects $\# >$ threshold
- Split into sub-hypercubes
- Rescaling vector $U$

**Mapping**
- Map data in the same sub-system into the same point $C_l = S_U(D)$
- Rescaling function

**Labeling**
- Label the data mapped to the point $C_l$ with the same value
- Label: $L = f_k(C_l)$, CPF distance
- Cantor Pairing Function

**Re-coordination**
- Sub system $\rightarrow$ home system
- Re-coordinate $D$: $D_r = (d_1 \left( \frac{d_1}{u_1} \right) \cdot u_1, ..., d_k \left( \frac{d_k}{u_k} \right) \cdot u_k)$
Example: 2D Point Data

- 7 points in a 2D space
- Each node can contain at most 2 points
Partition using $U=(3.0, 3.0)$

The hypercube $(0,0), (9,9))$ is split into sub-hypercubes.
Map points to $C_I$ (red points)

Each point can be mapped to the lowest corner point of the sub-hypercube.

7 points
$B = (0,0), (9,9)\)$
$U = (3.0, 3.0)$
Label using CPF

\[ L = f_k(C_l) \]

The points are labeled by the CPF distance of the red points.
Re-coordination

Points:
(4, 5) → (1, 2)
(5.5, 4.5) → (2.5, 1.5)
(4.2, 3.5) → (1.2, 0.5)

Sub-hypercube
(3, 3) → (0, 0)
(6, 6) → (3, 3)

\[ D_r = (d_1 - \left\lfloor \frac{d_1}{u_1} \right\rfloor \cdot u_1, \ldots, d_k - \left\lfloor \frac{d_k}{u_k} \right\rfloor \cdot u_k) \]
Recursive procedure

\[
U = (1.5, 1.5)
\]
2D Dataset & PL-Tree

Final PL-Tree Structure

7 points
B = (0,0), (9,9)
U = (3.0, 3.0)
E = (0, 3, 4, 18)

3 points
B = (3,3), (6,6)
U = (1.5, 1.5)
E = (0, 1, 4)

2 1 1
1 1 1
Indexing Algorithms

• **Static Data Indexing**
  – Recursive 4-steps process

• **Dynamic Data Indexing**
  – Searching: calculate and search the label recursively from the root to the leaf node
  – Insertion: invoke a searching algorithm and insert data into the leaf node
  – Deletion: invoke a searching algorithm and remove the found data from the leaf node; a merge process may be execute after a deletion
Similarity Searching Queries

- **Point query**
  - Searching algorithm

- **Range query**
  - Innerblocks
  - Outerblocks

- **k-Nearest-Neighbors (kNN) query**
  - Incremental algorithm: priority queue
  - Report one by one, in order of distance
Improvement & Optimization

• Compact PL-Tree Storage
  – “Curse of Dimensionality” \( \Rightarrow \) sparse distribution
  – One node one page \( \Rightarrow \) multiple nodes one page
  – Particular compact data structure

• Selection of rescaling vector \( U \)
  – PL-Tree is NOT balanced
  – Height Estimation:
    \[
    h \leq \log_{n} \rho \sqrt{k} + 1
    \]
  – BindWidth Optimization on each dimension

• Indexing Spatial Objects
  – Logical cut
  – Huge Object List
  – Range and kNN queries
Experimental Configurations

• **Environment**
  – Intel Core i5 2.53G
  – 4GB memory
  – Page size: 4KB (d<8) and 8KB (d>=8)
  – LRU buffer of 1000 pages

• **Datasets**
  – Synthetic Data: 1,000,000 points, 15D
  – Real-world Data
    • USPP: 15,206 points, 2D
    • LLMPP: 500,000 / 1,843,200 points, 15D
    • MAPS: 40,398 points, 39D
    • TIGER: 556,696 objects, 2D

• **Other Indices**
  – R*-trees and X-trees
  – Quadtrees
  – GridFile, S-Scan, and iDistance
The size of a PL-Tree index is constant to the dimensionality but that of a R-tree-based index increases linearly with dimensionality.

Synthetic data
D = 2, 3, 4, 5, 6, 8, 10, 12

PL-Trees v.s. R*-trees
D=2: 0.62 times
D=12: 0.08 times
Point Queries

• Synthetic data, 500,000 points, D=2,...,12
• Randomly select 10,000 points
  – The numbers of page accessed
  – The total elapsed time of queries
• PL-Trees outperform when the dimensionality is high
Range Queries

- **Performance over dataset size**
  - Selectivity = 0.1%, 10 random range queries
  - $D = 12$
  - Dataset size: from 100,000 to 1,000,000
- **Slightly better on larger datasets**

![Graphs showing performance over dataset size with different tree structures](Image)
Range Queries Cont.

- **Performance over data dimensionality**
  - Selectivity = 0.1%, 10 random range queries
  - $D = 3, 6, 9, 12$
  - Dataset size: 1,000,000 points
- **PL-trees maintain a smooth curve despite the dimensionality growth**
Range Queries Cont.

- **Performance over selectivity**
  - Selectivity: from 0.001% to 20%

- **Tiger Dataset**
  - 556,696 objects, 2D non-uniformly distributed
**kNN Queries**

- Compared with R*-trees only
- MAPS dataset, D = 2, 4, 6
- 1000 random kNN queries
- The number of pages accessed
- Superiority increases with dimensionality

![graphs](images)
OUTLINE

• Introduction
• Background
• Problem Definition
• Our Approach
• Future Work
• Conclusion
Future Work

• **Curse of Dimensionality**
  – Analyze high-dimensional data
  – Sparse data and compression

• **Optimization & Application of PL-Tree Indexing**
  – Selection of $U$
  – Ultra-high dimensionality
  – PL-Trees perform well for range queries in high-dimensional spaces
    • Traffic Moving Objects data
    • Web Term-Document data
  – Grid-based Clustering

• **Blessings of Dimensionality**
  – [Donoho, 2000]: Concentration of Measure, Asymptotic Distribution, and Approach to Continuum
Conclusion

• High-dimensional data trend
  – Curses and blessings of dimensionality

• Organization of high-dimensional data
  – CPF: bijective and invertible
  – CPU processing v.s. disk I/O
    • Fast calculation algorithm
    • Disk I/O saving algorithm

• Numerical results show PL-Tree indexing outperforms the existing indexing methods
References


Thank You

Questions?