VLDB 2020 Tutorial

Similarity Query Processing for High-Dimensional Data

Jianbin Qin
Shenzhen Institute of Computing Sciences
Shenzhen University

Wei Wang
University of New South Wales

Chuan Xiao
Osaka University and Nagoya University

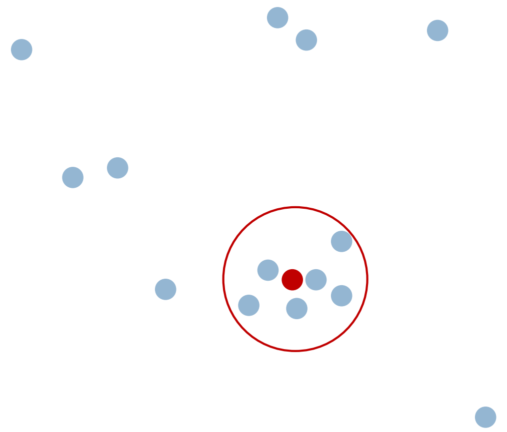
Ying Zhang
University of Technology Sydney

Outline

- Introduction
- Exact Query Processing
- Approximate Query Processing
- Selectivity Estimation
- Open Problems

Exact Query Processing

- Problem definition
  - Range-similarity query
    - Given:
      - a database *X* of high-dimensional vectors,
      - a query vector **q**,
      - a distance function *dist*(., .),
      - a threshold *t*.
    - Return **ALL** the objects **x** in *X* such that *dist*(**q**, **x**) ≤ *t*.
    - a.k.a. range-similarity query or t-selection problem
  - Given:
      - …
      - a number ***k***.
    - Return **ALL** the **k** objects R in X such that no other objects is closer to q than objects in R.
    - A.k.a. k nearest neighbor query



Motivation

- EXACT does not pose any uncertainty to the pipelines that apply similarity query processing as a component.
- It also simplifies empirical comparison as only speed and space consumptions are key evaluation criteria.
- Where is boundary of the exact and approximate query processing lies.

Challenge

- The curse of dimensionality
  - The computation of **exact NN** solution is very expensive.
  - Research effort has been attracted to **approximate NNS**.
    - Locality sensitive hashing (LSH)-based methods.
      - C2LSH, LSH-tree, SRS.
    - Product quantization (PQ)-based methods.
      - PQ, OPQ, LOPQ.
    - Neighborhood graph-based approaches.
      - KGraph, Small world Graph.

Opportunity

- Opportunity: the intrinsic dimensionality of real-life high dimensional data is usually much lower.
  - It is still feasible to develop efficient and practical exact NNS method.
  - Tree index-based method.
    - KD-tree, iDistance, Cover Tree.
  - Following the “filter and verify” paradigm.
    - PartEnum, HmSerach, MiH, GPH, Pigeonring.

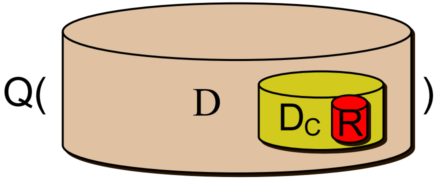
Outline

- Partitioning Methods. (Divide and conquer)
  - These methods partition the original space and bound the overall distance using the distance in each subspace.
- Dimensionality Reduction Methods
  - These methods project objects to another space to reduce dimensionality.
- Tree based methods (next part)
  - These methods partition the database in a hierarchical manner.

Partition based – Solve τ-selection Problem (Range Similarity Query)

Challenges:

- When D is large, straightforward searching is costly.
- D and f may be complex, and hard to be indexed directly.



General Solution: Divide and conquer

$$tS(D, Q, \tau) = Verify(tS(D_{(1)}, Q_{(1)}, \tau_1), tS(D_{(2)}, Q_{(2)}, \tau_2), \dots)$$

Step 1: Decompose f into several parts, such that $f_1(x_1, q_1) + f_2(x_2, q_2) + \dots + f_m(x_m, q_m) \leq \tau$

Step 2: Perform candidate generation, such that CAND = $Q_1(D_1, q_1, f_1, \tau_1) \cup Q_2(D_2, q_2, f_2, \tau_2) \cup \dots \cup Q_m(D_m, q_m, f_m, \tau_m)$.

Step 3: Verify x in CAND, such that $f(x, q) \leq \tau$

Multi-Index Search (PartEnum VLDB2004, HmSearch SSDBM2012, MIH CVPR2012….)

- **Reduction** via pigeonhole principle

Number of partitions:
$m = 3$

$$HS(D, Q, \tau) = Verify(HS(\mathrm{D}_{(1)}, Q_{(1)}, \tau_1), HS(\mathrm{D}_{(2)}, Q_{(2)}, \tau_2), \ldots)$$

$$\tau_1 = \tau_2 = \tau_3 = \lfloor \frac{\tau}{m} \rfloor$$

$DB_{(1)}$ $\tau_1 = 1$

1	1	0
1	1	1
1	1	1
1	1	1
1	1	1

1	1	1

$Q_{(1)}$

$DB_{(2)}$ $\tau_2 = 1$

0	1	0
0	1	1
1	1	0
1	0	0
1	1	0

0	1	0

$Q_{(2)}$

$DB_{(3)}$ $\tau_3 = 1$

0	0	0	1
0	0	0	0
0	1	1	0
1	1	1	0
1	1	1	0

0	0	0	1

$Q_{(3)}$

Naïve Pigeonhole Principle (ICDE12, SSDBM13, CVPR 2012)

- **Tightness of divided-thresholds**

$$\tau_1 = \tau_2 = \tau_3 = \lfloor \frac{\tau}{m} \rfloor$$

	τ_1	τ_2	τ_3
$\tau = 5$	1	1	1
$\tau = 4$	1	1	1
$\tau = 3$	1	1	1

→ **Same set of** candidates

Naïve Pigeonhole Principle (CVPR 2012)

- **Vulnerable to *data skewness***
  - Data skewness is quite common
- Most solutions to data skewness
  - Do nothing, or
  - Shuffle the columns, and then sequential partitioning. Hopefully each partition is less likely to be extremely skewed [SSDBM13, CVPR12]

DB:	1	1	0	0	1	0	0	0	0	1
	1	1	1	0	1	1	0	0	0	0
	1	1	1	1	1	0	0	1	1	0
	1	1	1	1	0	0	1	1	1	0
	1	1	1	1	1	0	1	1	1	0
Q:	1	1	1	0	1	0	0	0	0	1

$\tau_1 = 1$ $\tau_2 = 1$ $\tau_3 = 1$

- All records in 1st partition are candidates →
- Verification for the entire DB, *irrespective of* other partitions

Achieve **Tight** Threshold Allocations General Pigeonhole Principle (GPH ICDE 2018)

- General Pigeonhole Principle
  - Allocate **different** thresholds to partitions
  - As long as the thresholds sum up to $\tau - m + 1$
  - Can be shown to be the tight
- $\tau_i \in \{-1, 0, 1, \ldots, \tau\}$
  - "-1" to allow discarding the partition
  - Correct and is the key to handle extreme skewness

$$\tau = 3$$

MIH thresholds:

$$\tau_1 = \left\lfloor \frac{\tau}{m} \right\rfloor = 1 \quad \tau_2 = \left\lfloor \frac{\tau}{m} \right\rfloor = 1 \quad \tau_3 = \left\lfloor \frac{\tau}{m} \right\rfloor = 1$$

GPH thresholds:

$\tau_1 = 0$	$\tau_2 = 0$	$\tau_3 = 1$
$\tau_1 = -1$	$\tau_2 = 0$	$\tau_3 = 2$
$\tau_1 = -1$	$\tau_2 = 1$	$\tau_3 = 1$

Adaptive Threshold Allocation (ICDE 2018)

- **Which threshold allocation is the best?**
  - Cost function:
    - Total number of candidates from the partitions
    - It upper bounds the query cost (up to some constant)
- Assumption:

  $CN(Q_i, u) \triangleq |HS(DB_{(i)}, Q_{(i)}, u)|$

  can be estimated $\forall i, u$
  - Use histogram, or
  - Use Machine Learning models

DB:

1	1	0	0	1	0	0	0	0	1
1	1	1	0	1	1	0	0	0	0
1	1	1	1	1	0	0	1	1	0
1	1	1	1	0	0	1	1	1	0
1	1	1	1	1	0	1	1	1	0

$< q_1, \tau_1 >$ $< q_2, \tau_2 >$ $< q_3, \tau_3 >$

Q:

1	1	1	0	1	0	0	0	0	1

$\tau = 3$ $d = 10$

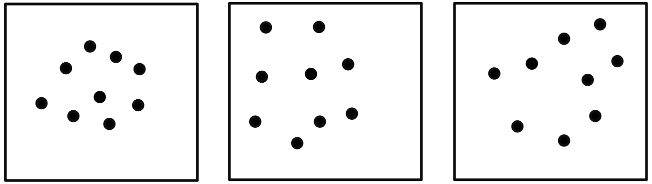
Minimize $CN(Q_{(1)}, u_1) + CN(Q_2, u_2) + CN(Q_{(3)}, u_3)$

Encourage Skewness (GPH ICDE 2018)

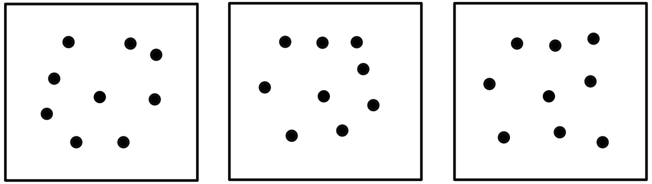
- **Let's make partitions more skewed !!**
  - Initial dimension partitioning
    - Greedy algorithm to minimize the total entropy of partitions
  - Refinement by local rearrangement
    - Move one dimension to another partition if it reduces the query cost

Dynamic Dimension Reduction

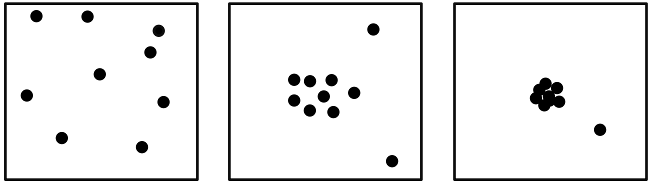
Origianl Data Partition



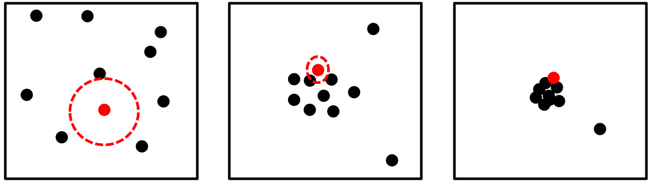
Random Shuffle Dimentions



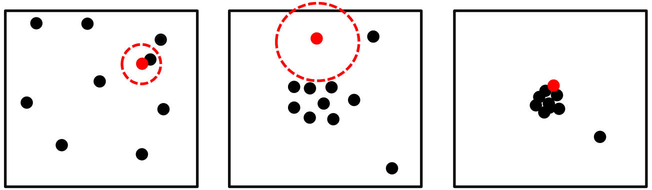
Skewnized Data Partition



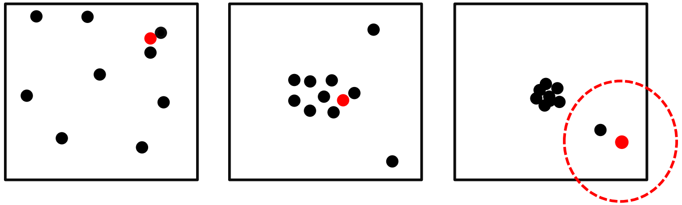
Query Q1, Allocate 1, 0 -1



Query Q2, Allocate 0, 1, -1

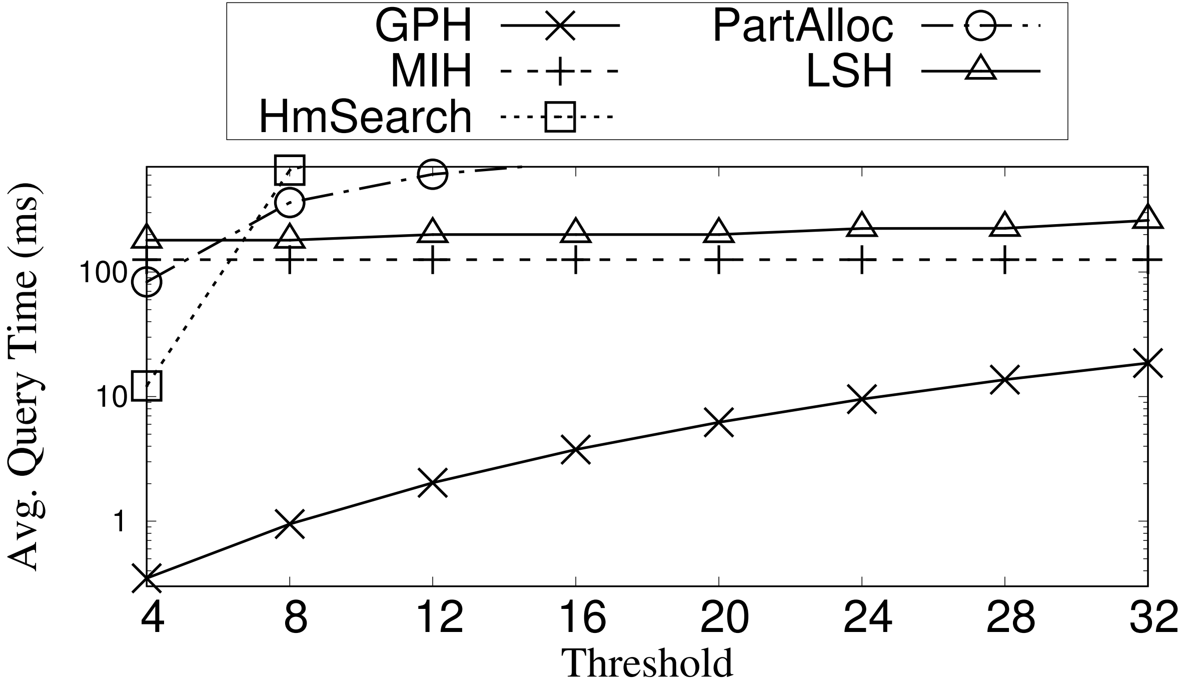


Query Q3, Allocate -1, -1, 2



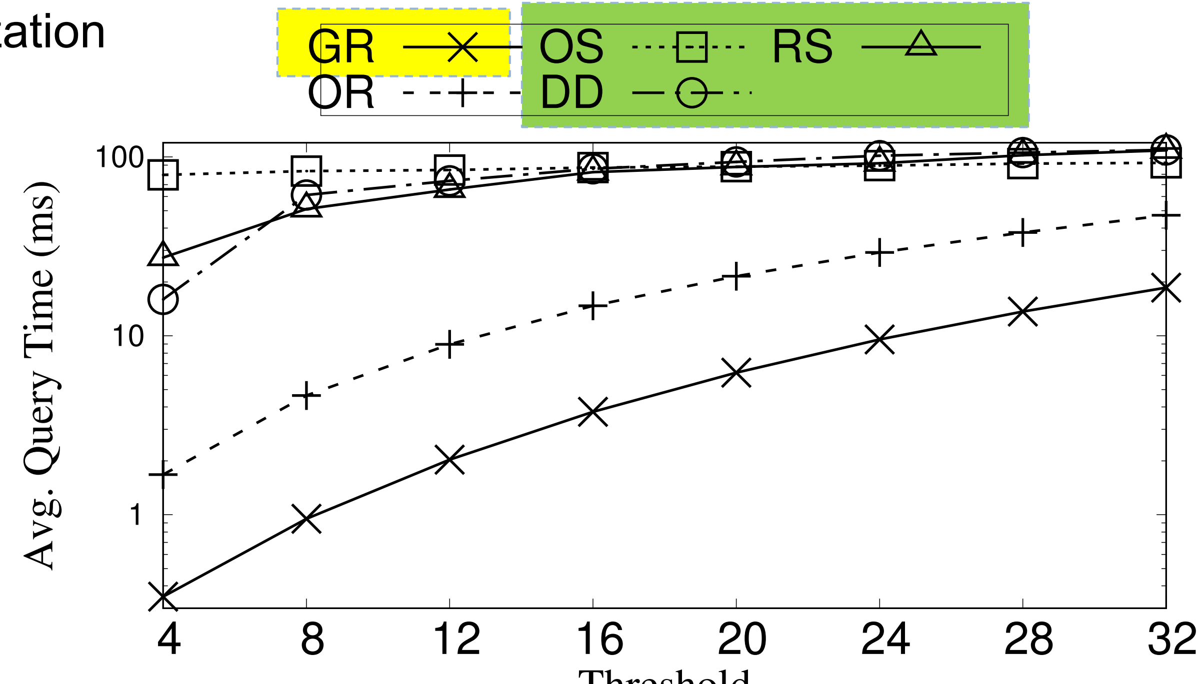
GPH Experiments - Running Time /2

- **PubChem dataset**
  - highly skewness ➔ existing methods lose their pruning power quickly

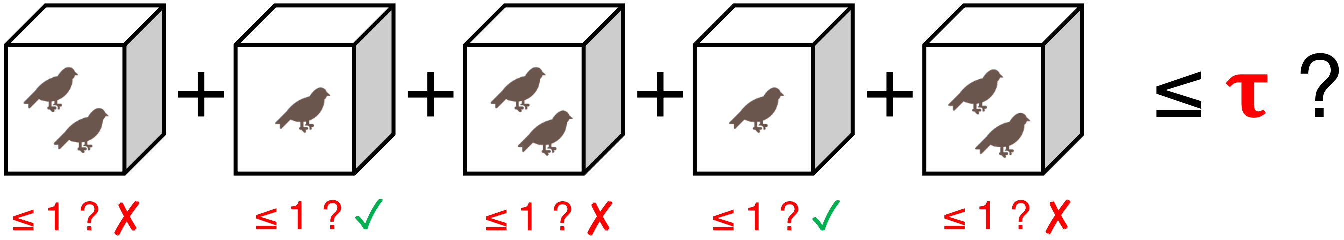


GPH Experiments - Dimension Partitioning (PubChem)

- OR: original dataset
- DD, OS, RS: existing methods that avoid skewness
- GR: Skewnization



Pigeonhole Principle (Multiple Boxes)



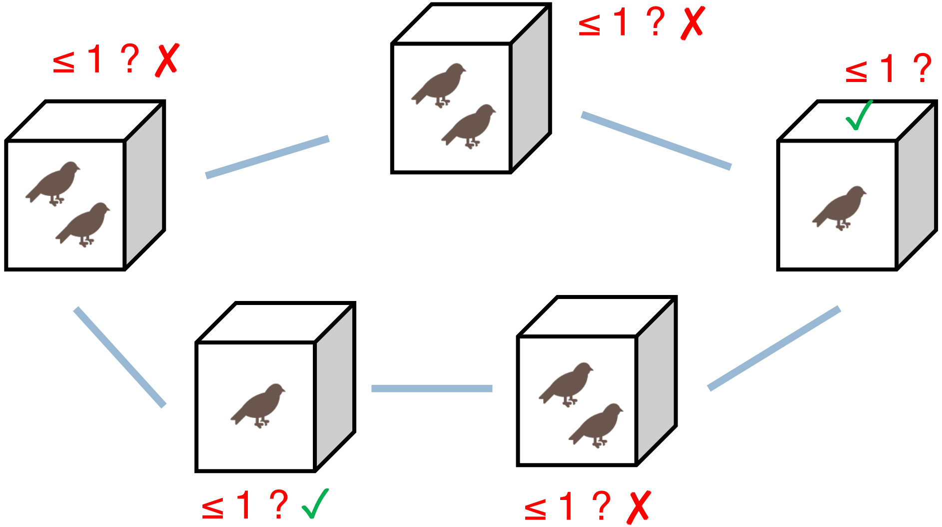
Basic Idea: Bound Multiple Boxes?

Problems: Exponential number of pigeonhole combinations.

- 20 combined 2 pigeonholes.
- 60 combined 3 pigeonholes.
- …

Pigeonring Principle: Basic form (VLDB19)

Dose m pigeonholes contain no more than τ pigeons?



$l = 1$

≤ 5 ?

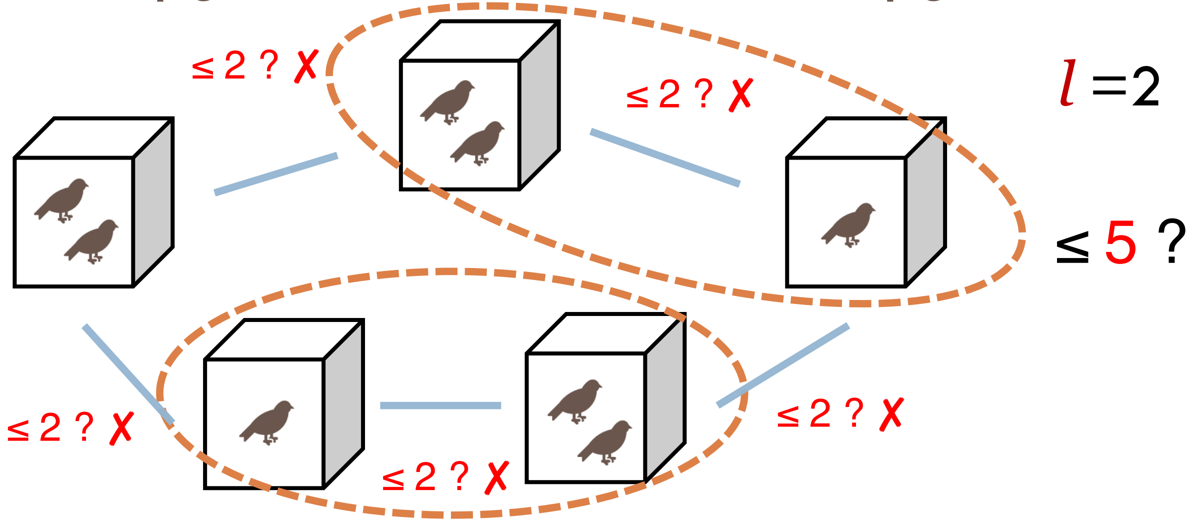
- Consider the adjacent partitions
- When l = 1, it is the same as General Pigeonhole Principle.

Define an order: Boxes are placed in a ring.

For every ***l*** in [1 .. m], there exist *l* consecutive boxes which contain a total of **no more than *l*·τ/*m*** pigeons.

Pigeonring Principle: Basic form. (VLDB19)

Dose m pigeonholes contain no more than τ pigeons?

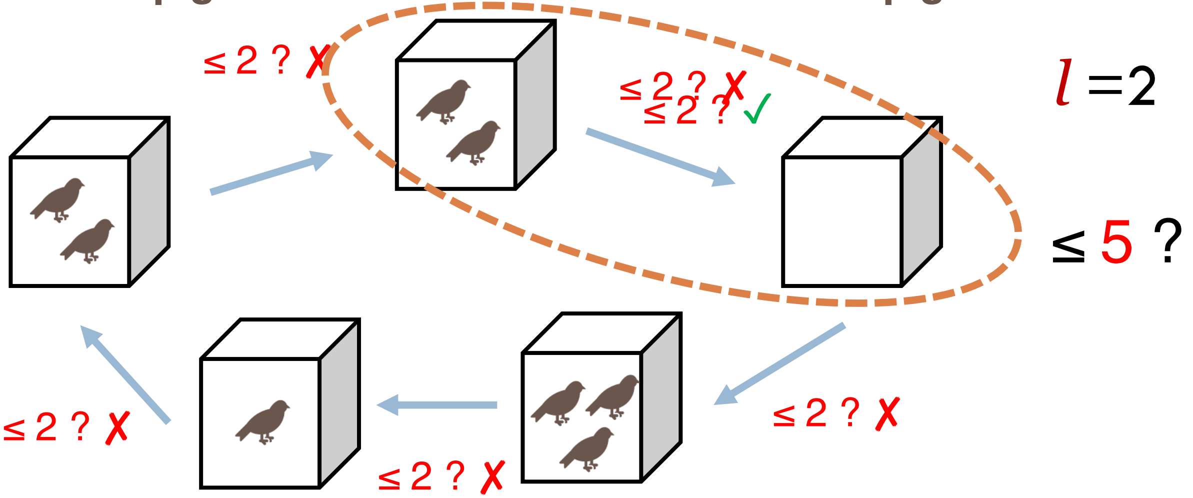


- Consider the adjacent partitions
- When l = 2, it is tighter than General Pigeonhole Principle.
- The record can be filtered!

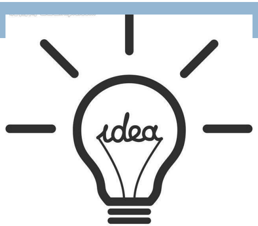
Define an order: Boxes are placed in a ring.
For every ***l*** in [1 .. m], there exist *l* consecutive boxes which contain a total of **no more than *l*·τ/*m*** pigeons.

Pigeonring Principle: Strong form (VLDB19)

Dose m pigeonholes contain no more than τ pigeons?



Add a direction, i.e., going clockwise.

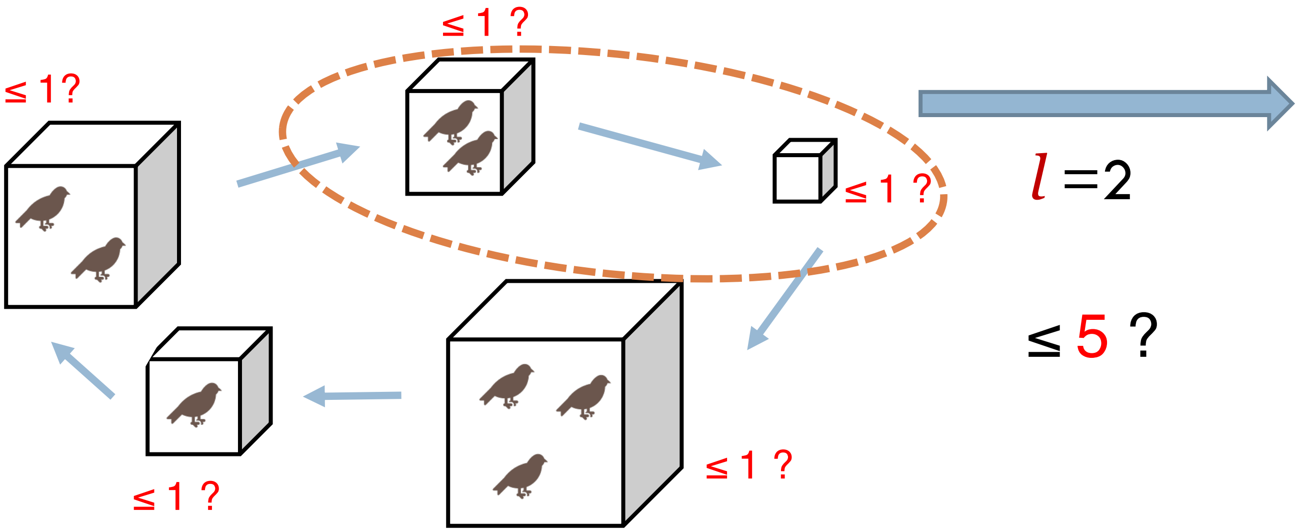


- Consider the adjacent partitions
- When l = 2, it is tighter than General Pigeonhole Principle.
- The record can be filtered!

There exists a pigeonhole such that for **every** l in [1 .. m], starting from this pigeonhole and going clockwise, the l consecutive pigeonholes contain a total of no more than $l \cdot t/m$ pigeons.

Combine with GPH Threshold Allocation (VLDB19)

Dose m pigeonholes contain no more than τ pigeons?



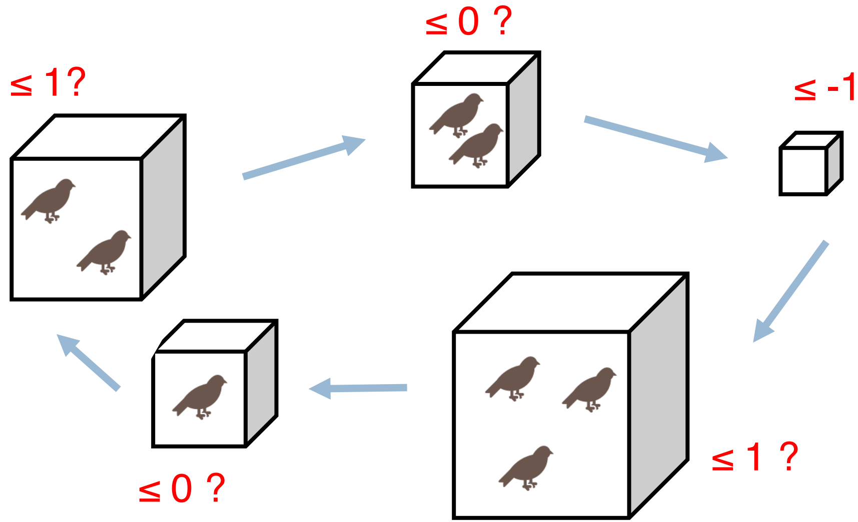
- Allocate 2 pigeons for the two holes
- Due to the non-uniform distribution of pigeons, even allocation is not good.

Not every pigeonhole is equal. **Non-uniform distribution.** i.e. Prefix Filtering

- Weak threshold allocation: every pigeonhole has equal τ/m partial threshold.
- GPH threshold allocation: We use an allocation vector $T = [\tau_0, \tau_1, \ldots, \tau_{m-1}]$.
  - Requires: $||T||_1 \geq \tau - m + 1$

Combine with GPH Threshold Allocation

Dose m pigeonholes contain no more than τ pigeons?



$l = 2$

≤ 5 ?

Pigeonring Principle +
GPH threshold allocation

Minimiz e

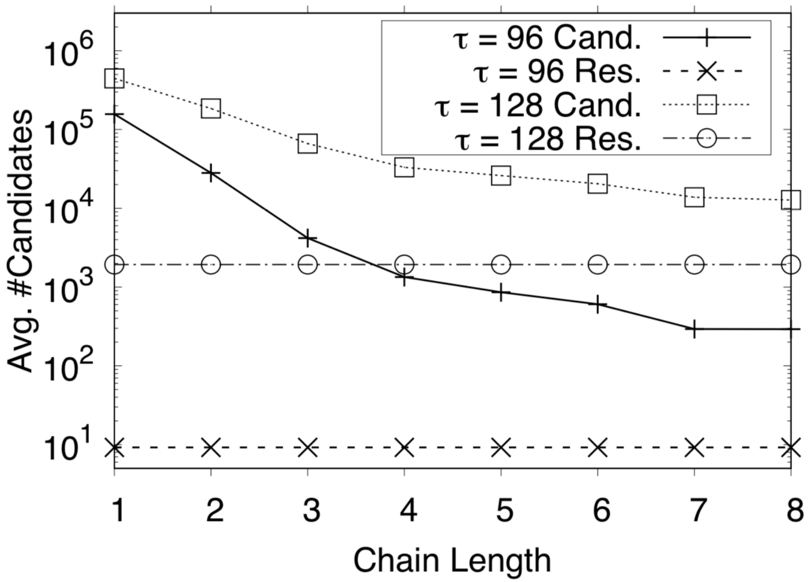
$$CN(q_1,\tau_1)+CN(q_2,\tau_2)+CN(q_3,\tau_3)$$

$$OPT[i,t]=\begin{cases}\min\limits_{e=-1}^{t+i-1} OPT[i-1,t-e]+CN(q_i,e) & \text{if } i>1\\ CN(q_i,t) & \text{if } i=1\end{cases}$$

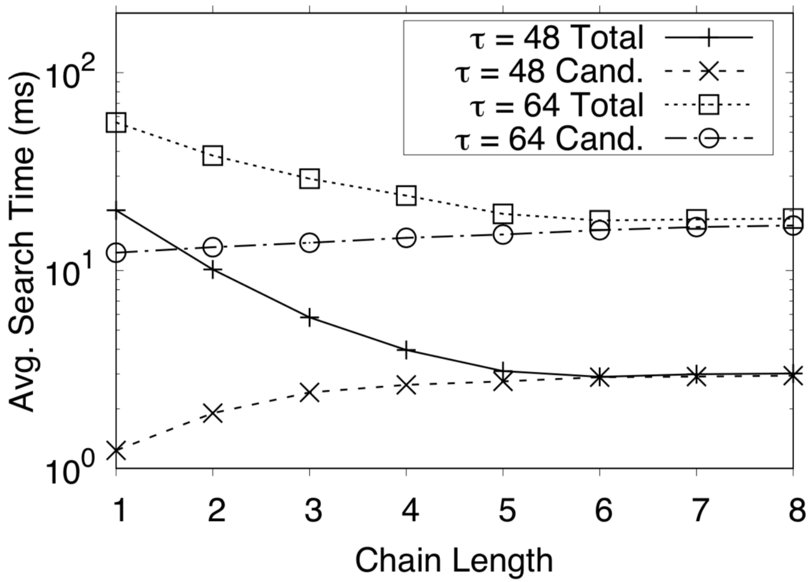
Not every pigeonhole is equal. **Non-uniform distribution.** i.e. Prefix Filtering

- Weak threshold allocation: every pigeonhole has equal τ/m partial threshold.
- GPH threshold allocation: We use an allocation vector $T = [\tau_0, \tau_1, \ldots, \tau_{m-1}]$.
  - Requires: $||T||_1 \geq \tau - m + 1$

Pigeonring – Experiment Study



(a) GIST, Candidate

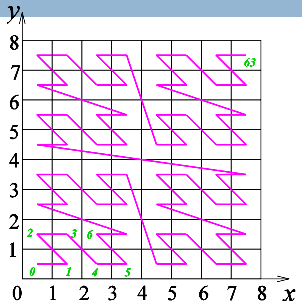


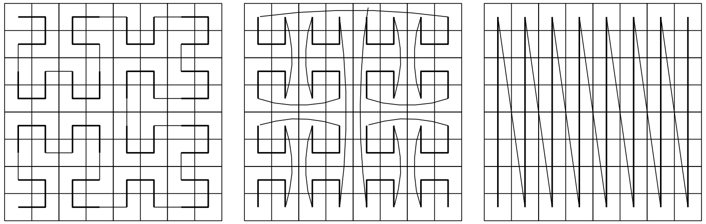
(b) GIST, Time

Effect of Chain Length on Hamming Distance Search

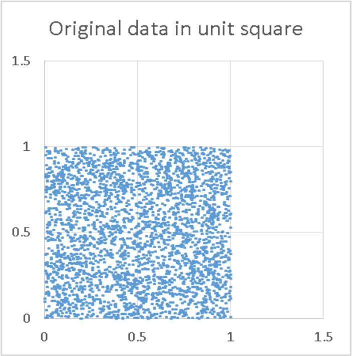
Other Dimension Reduction Based methods

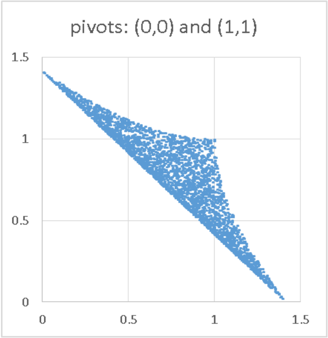
- Space Filling Curve
  - Not work for high

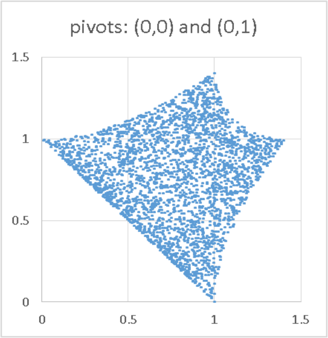




- Metric Space index (Pivot selection)







Neighboring corners are better than opposite corners!

Embedding Method with Guarantee (DASFAA 2018)

- An efficient distance lower bound
  - use the combination of linear and non-linear embedding.
- Dimensionality reduction
  - each point in a high dimensional space is embedded into a low dimensional space .
- Following “*filter-and-verify*” paradigm
  - develop an efficient exact NNS algorithm by pruning candidates using the new lower bounding,
  - hence reducing the cost of expensive distance computation in original space.

Summary of the Exact Techniques

Index	Disk-based / In-memory	Efficient query type	Dimensionality	Comments
R-tree	Disk-based	Point, window, kNN	Low	Disadvantage is overlap
K-d-tree	In-memory	Point, window, kNN(?)	Low	Inefficient for skewed data
Quad-tree	In-memory	Point, window, kNN(?)	Low	Inefficient for skewed data
Z-curve + B^+-tree	Disk-based	Point, window	Low	Order of the Z-curve affects performance
iDistance	Disk-based	Point, kNN	High	Not good for uniform data in very high-D
VA-File	Disk-based	Point, window, kNN	High	Not good for skewed data
GPH	Memory-based	Range, KNN	High	Good for Skewed data
Pigeonring	Memory-based	Range	High	Good for Skewed data
LNL	Disk-based	KNN	High	Good for Skewed data

Thank You!

Q & A