KD-Tree Implementation and Many Dataset Customization

Erik Hasprunár
396122@mail.muni.cz

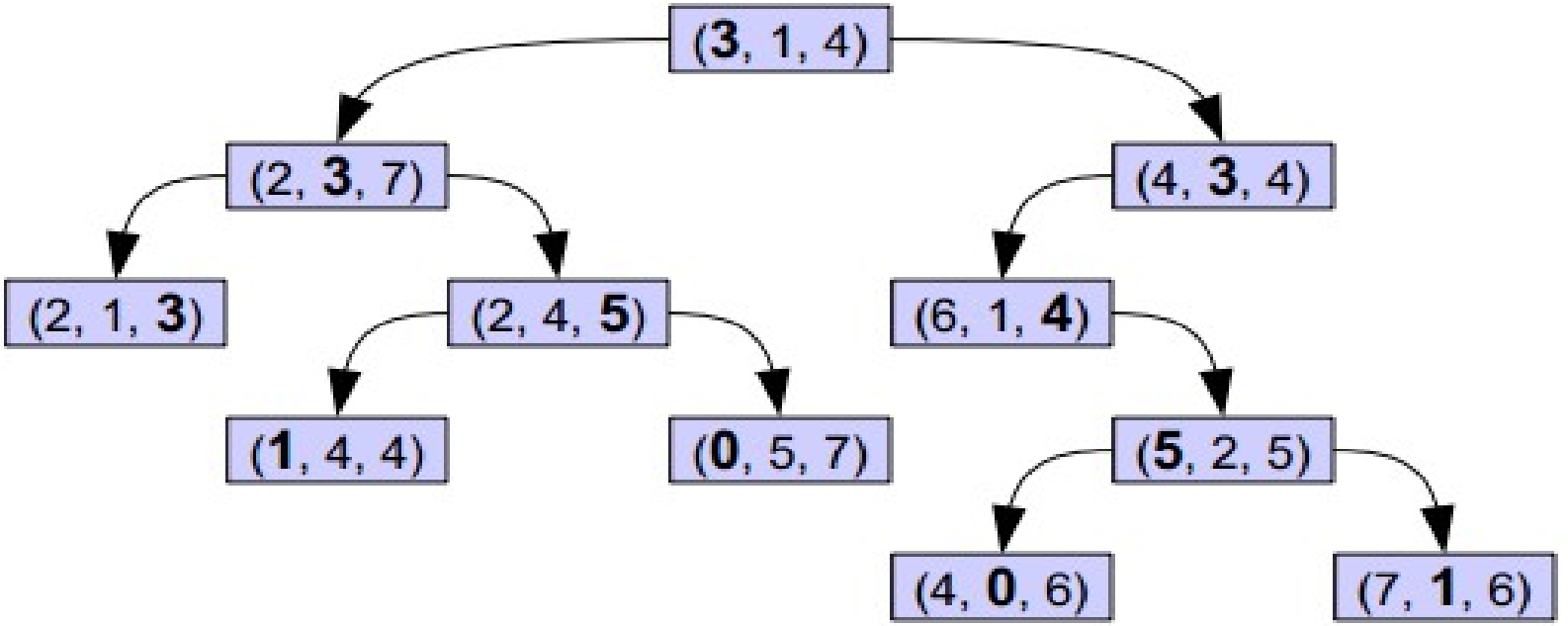
Motivation & Objectives

- Motivation:
  - Similarity searching
    - Computer vision(e.g. bag-of-words model)
- My (bachelor) work:
  - Implementation of KD-tree
    - to MESSIF library(Java)
  - Indexing many datasets together

KD-Tree

- Space-partitioning data structure
- **Binary** tree
- Splits k-dimensional data each **node** based on **one chosen dimension**

KD-Tree



- Example on 3-D data
- Number in **bold** is the chosen **split dimension**

Building KD-Tree

- Built recursively
- Choosing for every node:
  - **split dimension (dim)**
  - **split threshold (t)**
- Splits data info **left** and **right** subtree
  - Lower data into **left** subtree (item[dim] < t)
  - All others into **right** subtree (item[dim] >= t)

Split Dimension

- Originally, split dimensions were chosen ciclically(1, 2, 3, 1...)
- In our implementation, we use VLFeat's approach(**VLFeat.org**)
  - Decision based on **variance** in all dimensions over all data
  - One of the few **highest-variance** dimensions is chosen at random and set as **split dimension**

Split Threshold

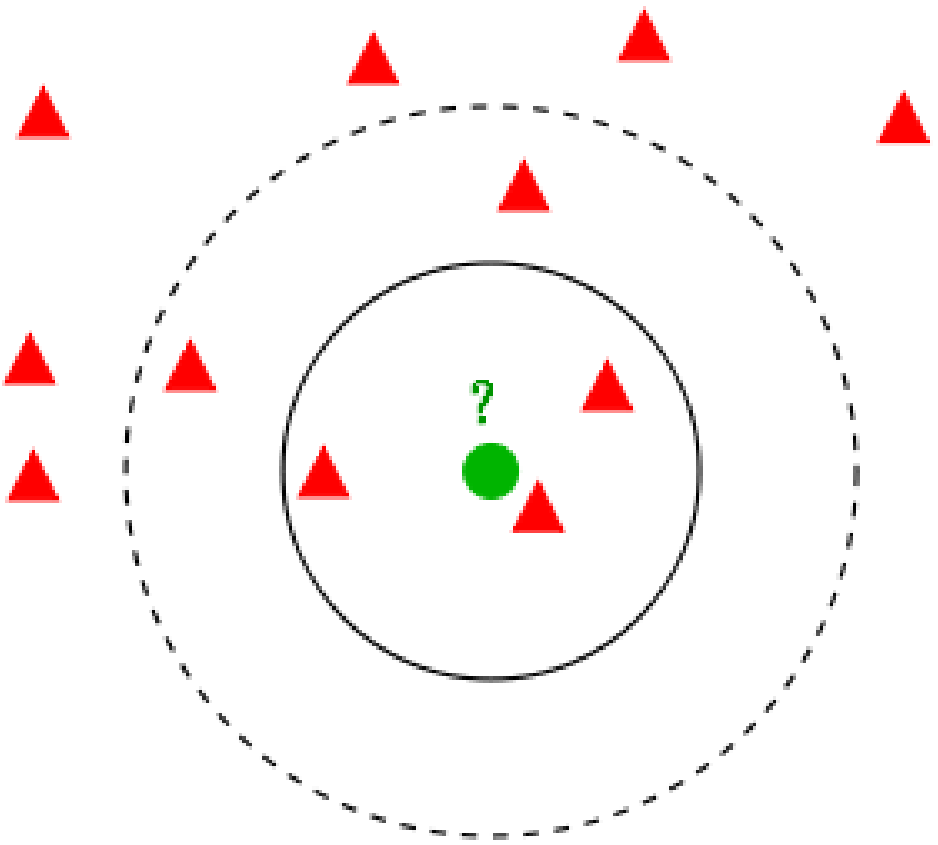
- Can be **mean** or **median**
- **Median** approach is used by default
  - balanced tree
- Data are then **sorted** by comparing the **split dimension** values and are **split** into subtrees

Leaf Node

- Only 1 datum given to the next node → datum is saved in the leaf node
  - Last level of nodes
  - After all leaf nodes are created, building of the tree ends

Searching in KD-Tree

- **k-NN query** (k-nearest neighbors) is implemented
  - **Green dot** = query
  - **Triangles** = data
  - **Solid line** circle
    - k = 3
  - **Dashed line** circle
    - k = 5



k-NN search

- Searches **recursively**
- **Priority queue** of nodes is handled
- **Lower-bounds** are computed for each node in the queue
  - minimal possible-distance answer that can be found in the node's subtree
- Search prioritizing based on the node's **split dimension**
  - **query[split dimension] < split threshold** → left child node is next, right child node is put into queue and vice versa

Leaf Node

- When the **leaf node** is reached, distance between the **query** and the **datum** in the leaf is computed
- Distance is compared to the highest-distance answer (found until now)
  - If (distance > highest-answer distance) → vector is not inserted
  - If (distance < highest-answer distance) → vector is inserted and the highest distance answer is thrown out
- After this, next node from priority queue is taken for next searching etc.

Stop condition

- If (highest-distance of answers < lowest-bound of nodes) → the search ends
- Stop condition is hardly achieved when the data are of high dimensionality
  - **“Curse of dimensionality”**
  - Can be solved by approximation

Approximation

- As the **distance computation** is the **most costly** operation, **maximum number** of these can be preset
- Choosing the right number is a problem yet to be solved(depends on how accurate the answer is needed, number of data, dimensionality etc.)

Randomized KD-Forest

- Same principle as KD-tree, but **more trees**
- Build:
  - As the split dimensions are chosen at random(partly), all trees are built differently
- Search:
  - One priority queue of nodes handles all trees(“jumping between trees”)
- Randomized forests have been found to be effective at approximate searches

Many Dataset Experiment

- Experiment motivation:
  - Build a tree with more independent datasets
    - Datasets share the same data space
  - Searching for k answers in every dataset (k * number of datasets = number of all answers)

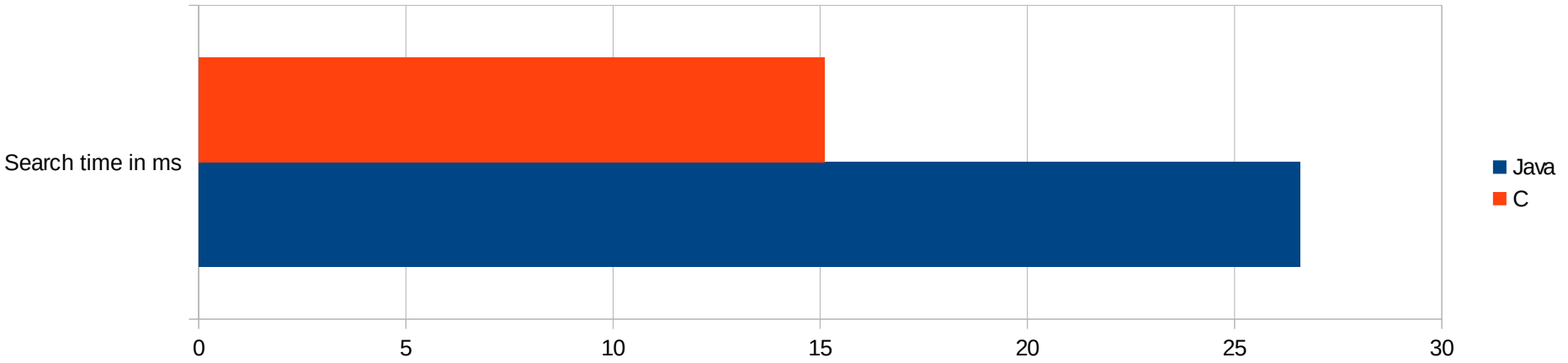
Many Dataset Customization

- Build:
  - Same as regular KD-tree
    + **Dataset id** is remembered for each data
- Search:
  - Same as regular KD-tree
    + Answers for every dataset are separate
    + Stop condition is calculated based on the highest-answer distance from **all** datasets

Testing System Specifications

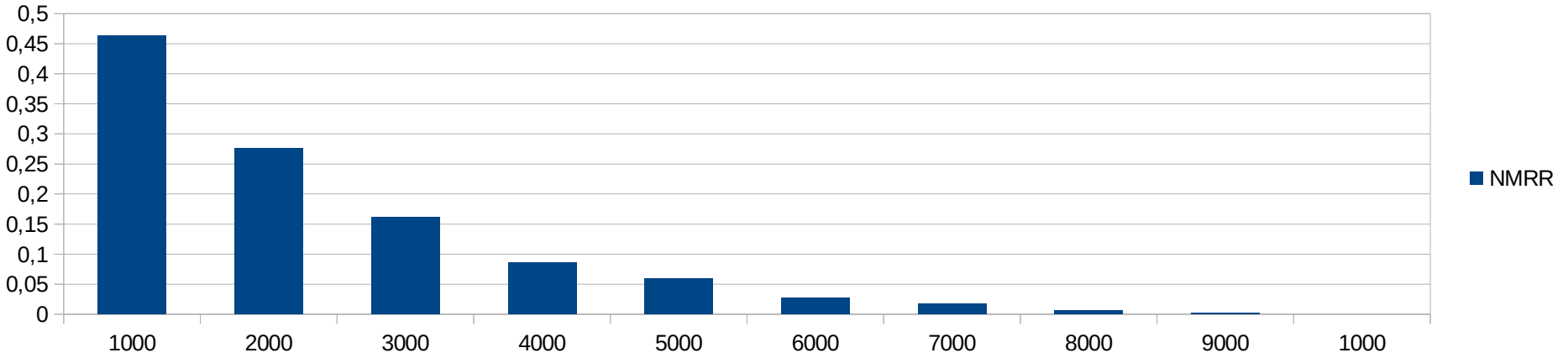
- Processor: AMD A4-3310mx
- OS: Win7 64-bit
- RAM: 4GB(3,48GB)
- IDE:
  - Netbeans 8.0.1(Java)
  - Microsoft Visual C++ 2008 Express Edition(C)

Mine(Java) vs VLFeat(C)



- Number of objects: 10 000
- K: 100
- Dimensionality: 128
- Number of trees: 1
- **10 000 Distance Computations for both**

Approximation Test



- Number of objects: 10 000
- K: 200
- Dimensionality: 128
- Number of trees: 1

Many Dataset Results

- No real statistics (sorry)
- Distance computations on low-dimension data looks fine
- Operation time looks bad for now (possible bad implementation)