Enhancing Similarity Search Throughput by Dynamic Query Reordering

Filip Nálepa, Michal Batko, Pavel Zezula
Faculty of Informatics, Masaryk University, Brno, Czech Republic

Big Data Processing

- Large amount of data produced every second
- Need to process the data
- Two basic approaches:
  - Store and process later, i.e., database processing
  - Process continuously, i.e., **stream processing**
- Examples of stream processing applications:
  - Surveillance camera stream and event detection
  - Mail stream and spam filter
  - Publish/subscribe applications

Stream Processing Scenarios

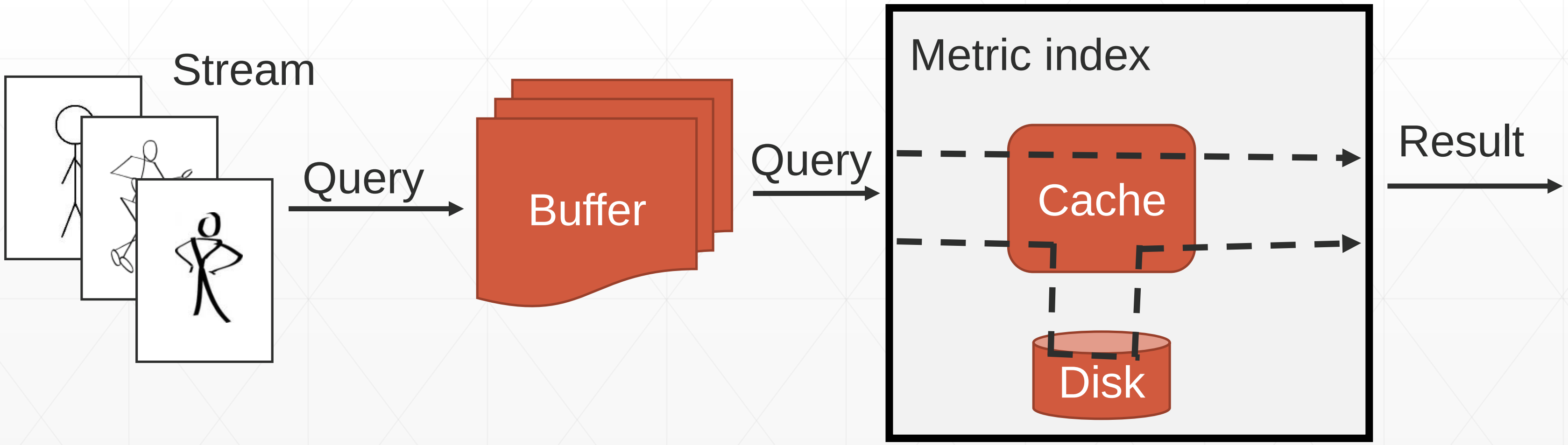
- Stream: potentially infinite sequence of data items (d_1, d_2, …)
- Basic scenarios:
  - Data items processed immediately, possible data item skipping → minimize delay
    - E.g., event detection in surveillance camera stream
  - Process everything as fast as possible, data item can be delayed → **maximize throughput**
    - That‘s our focus
- Motivating examples in similarity search
  - Image annotation – annotate a stream of images collected by a web crawler
  - Publish/subscribe applications – categorize a stream of documents
  - → stream of query objects

Problem Definition

- Domain of objects D
- DB of objects D indexed in the metric space
  - Distance function d: D x D → **R** determines the similarity of two objects
- Stream of query objects $((q_1, t_1), (q_2, t_2), \dots)$
  - $q_i \in D$
  - t_i – time of arrival, $t_i \leq t_{i+1}$
- Evaluate k-NN query for each q_i, i.e., find k most similar objects in DB to q_i
- Optimization criteria – throughput
  - Maximize the number of processed query objects

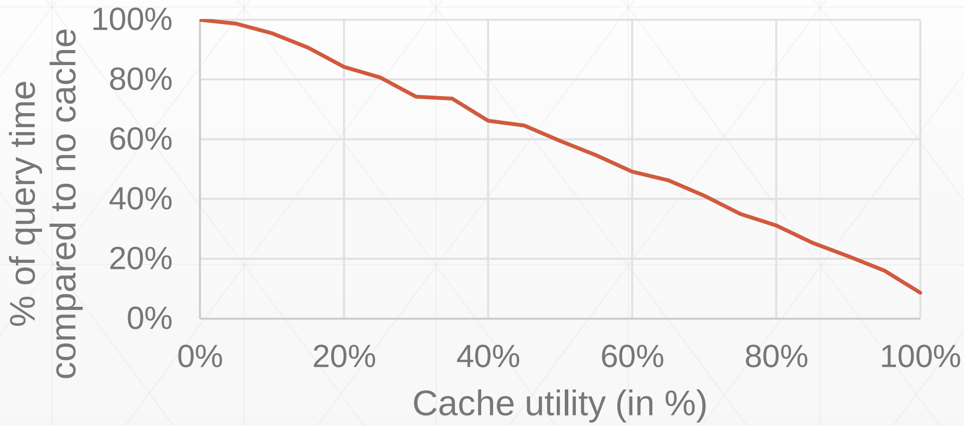
Architecture

- Typical similarity search techniques:
  - Partitioned data of DB stored on a disk
  - Read a subset of partitions during query evaluation → bottleneck
- Idea: similar query objects need similar sets of partitions → save disk accesses
- Buffer: waiting query objects, query object reordering
- Metric index: query evaluation
- Cache: in-memory caching of data partitions



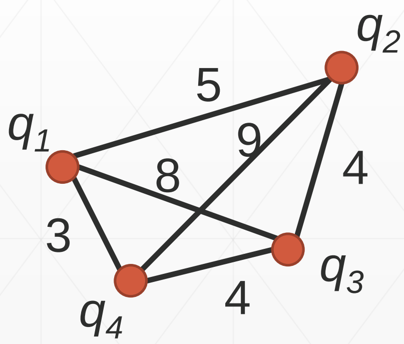
Cache

- Generic metric index
  - Data partitioning $P = \{p_1, \ldots, p_n\}$ where $p_i \subseteq D$
  - $I(q) \subseteq P$; partitions accessed during evaluation of q
- Partitions caching
- $cache = \{p_1, \ldots, p_m\} \subseteq P$
- Cache utility $cu = \frac{|I(q) \cap cache|}{|I(q)|}$
- Time to process a given query: *queryTime*(*cu*)
- Assumption: $cu_1 \leq cu_2 \rightarrow queryTime(cu_1) \geq queryTime(cu_2)$



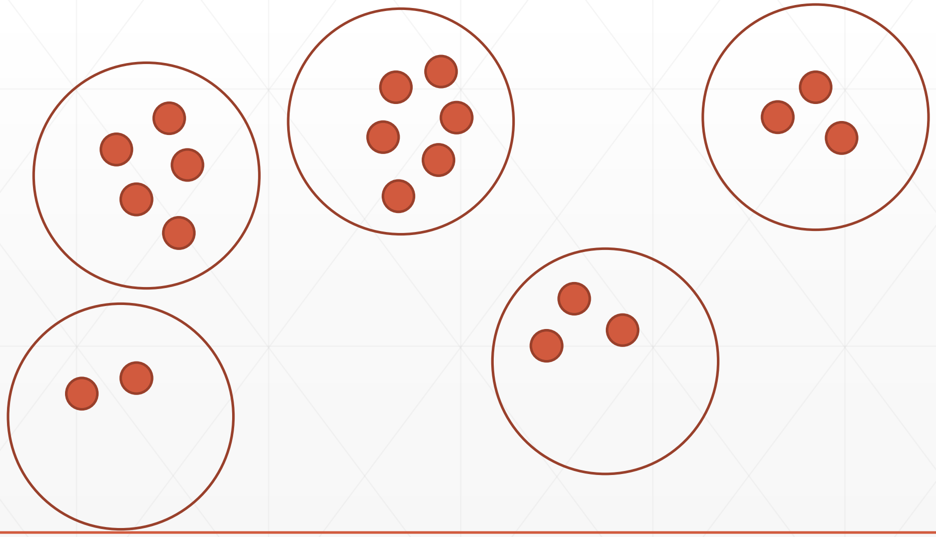
Buffer – Query Ordering

- Simplified buffer representation as an undirected complete graph G
  - Vertices = query objects in the buffer
  - Value of edge |*pq*| = time to process *q* after *p* (depends on the cache utility)
- Query ordering = path in G
- Throughput maximization: shortest path in G
- How to find a short path?
- How to construct the graph?



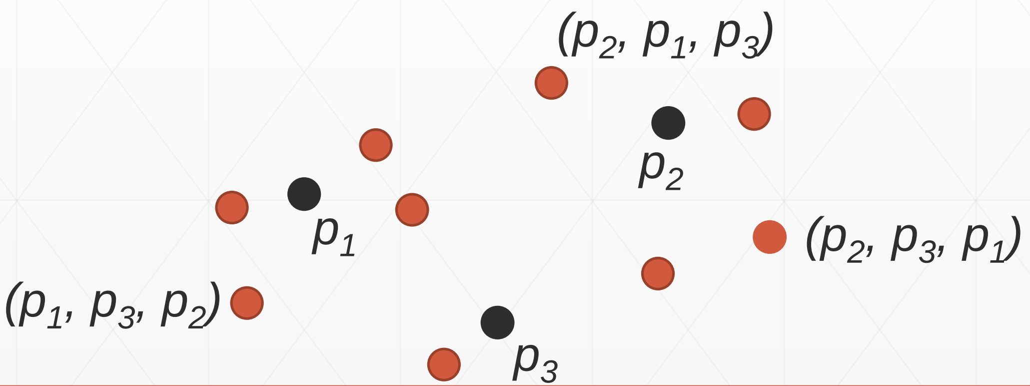
How to Find a Short Path?

- Shortest path search – NP-hard problem (travelling salesman)
- Added difficulty: new vertices added dynamically as new query objects arrive to the buffer
- Heuristics: find a dense cluster and evaluate queries in the cluster



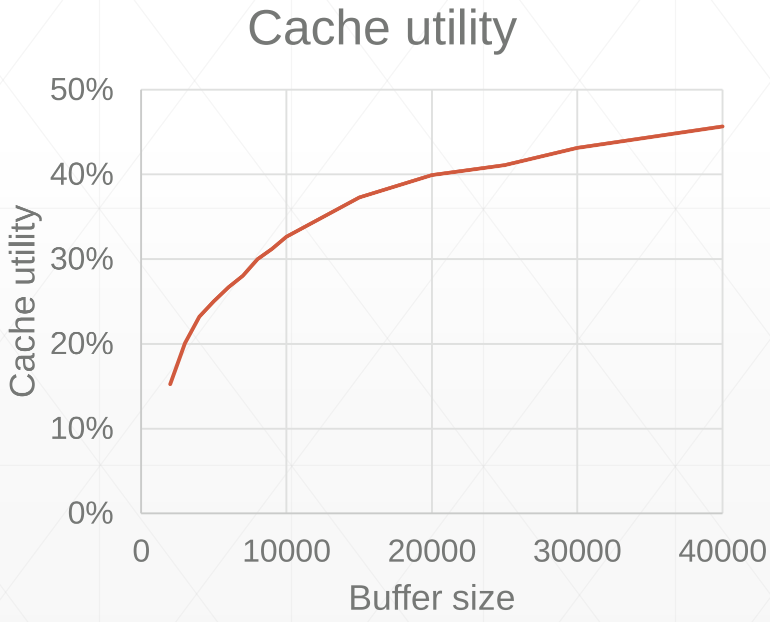
How to Construct the Graph and the Clusters?

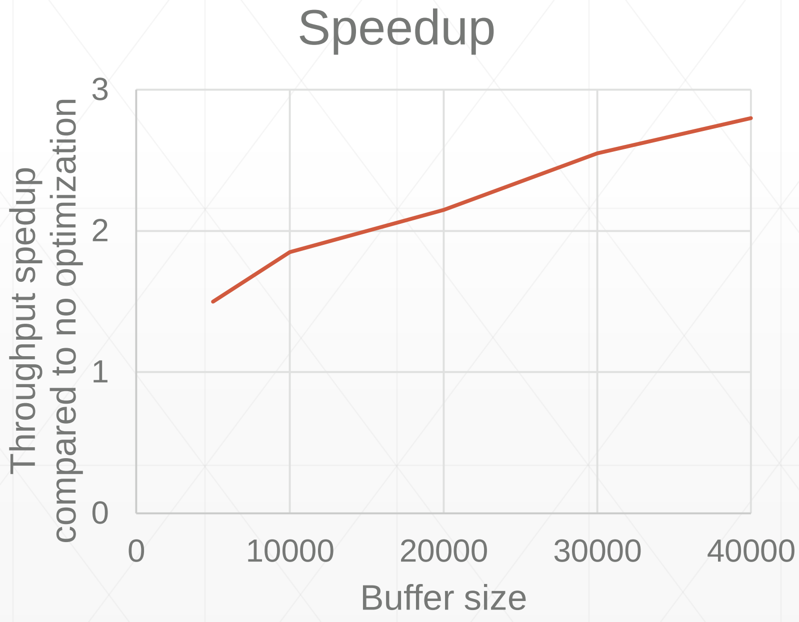
- Requirements: efficient, support for graph evolution
- Approach: estimate the edge values (query times) by metric distances
  - Low metric distance → high cache utility → low query time
- Computing all metric distances: time consuming
- → **Pivot-based clustering**
- Fixed set of pivots $p_1, \ldots, p_n$ in the metric space
- Compute metric distance of a new query object to all the pivots
- Order the pivots from the nearest to the farthest one → pivot permutation = cluster



Experiments – Fixed Buffer Size

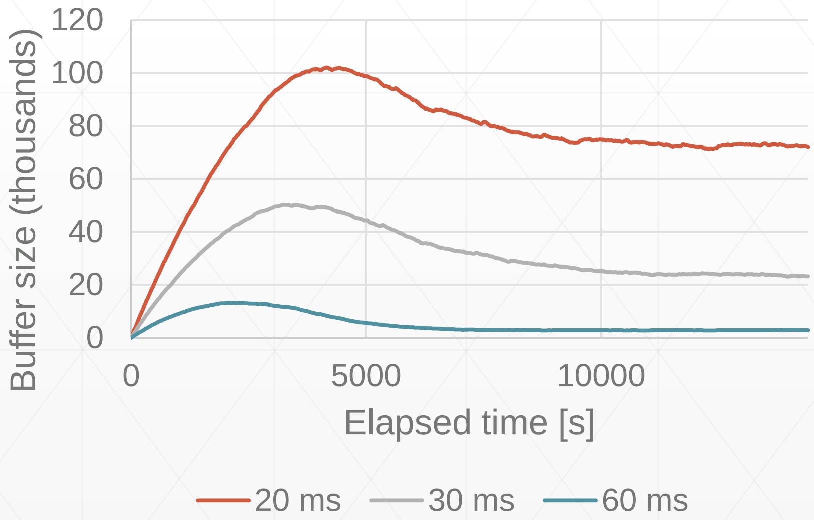
- DB: 10 mil. images represented by MPEG-7 descriptors
- Stream of query objects: evaluation of approximate 10-NN queries
- Cache size: 90,000 objects (0.9% of the DB)
- Fixed buffer size: 1 query object added per 1 processed query
- Baseline: no reordering, no caching





Experiments – Fixed Input Rate

- DB: 10 mil. images represented by MPEG-7 descriptors
- Stream of query objects: evaluation of approximate 10-NN queries (10 nearest neighbors)
- Cache size: 90,000 objects (0.9% of the DB)
- Fixed input rate: new query object arrives every x time units
- Average query time for no reordering and no caching: 113 ms



	20 ms	30 ms	60 ms
Max delay [s]	4031	2988	1565
Median delay [s]	1525	894	234
Cache utility	0.78	0.59	0.30

Delay: time since query object arrived until it was processed

Summary

- Stream of similarity query objects
- Enhancing the throughput by query reordering and data partition caching