Large Scale Content-based Image Retrieval using Standard Text Retrieval Engines

Claudio Gennaro

claudio.gennaro@isti.cnr.it

Outline

- Surrogate Text Representation
- Global features
- Vector of Locally Aggregated Descriptors
- Deep Convolutional Neural Network Features

Text Retrieval Engines for CBIR

- Bag of features represents probably one of the first attempt to exploit a Text Retrieval Engine for Content-based Image Retrieval.
- This transformation describes an image with a list of so-called visual words by extracting local visual features and quantizing them to decrease their variability.
- The great advantage of such an approach is that an efficient implementation through inverted files, e.g. Apache Lucene search engine, is available.

Credit: Dohnal, Vlastislav, Tomas Homola, and Pavel Zezula. "MDPV: metric distance permutation vocabulary." *Information Retrieval Journal* 18.1 (2015): 51-72.

Poorly annotated archives

- A person produce about 1000-2000 untagged pictures per year
- It is becoming difficult to retrieve both images published on the web and stored on personal computers
- Flickr:
  - 90% images do not have comments or tags

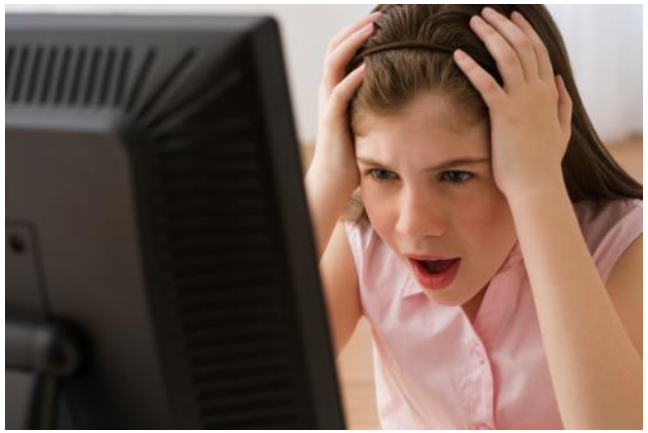


Image Similarity: *query-by-example*

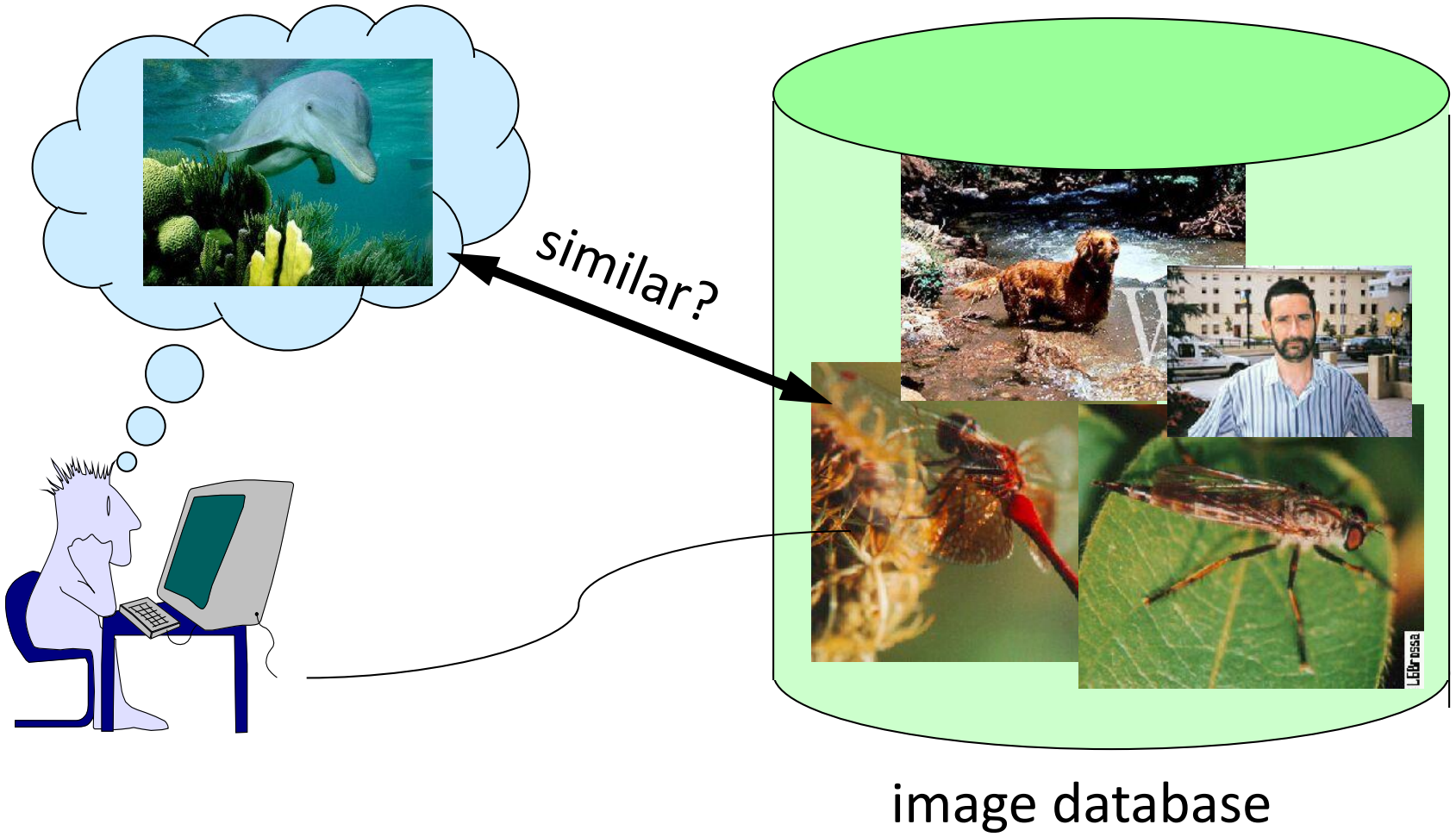
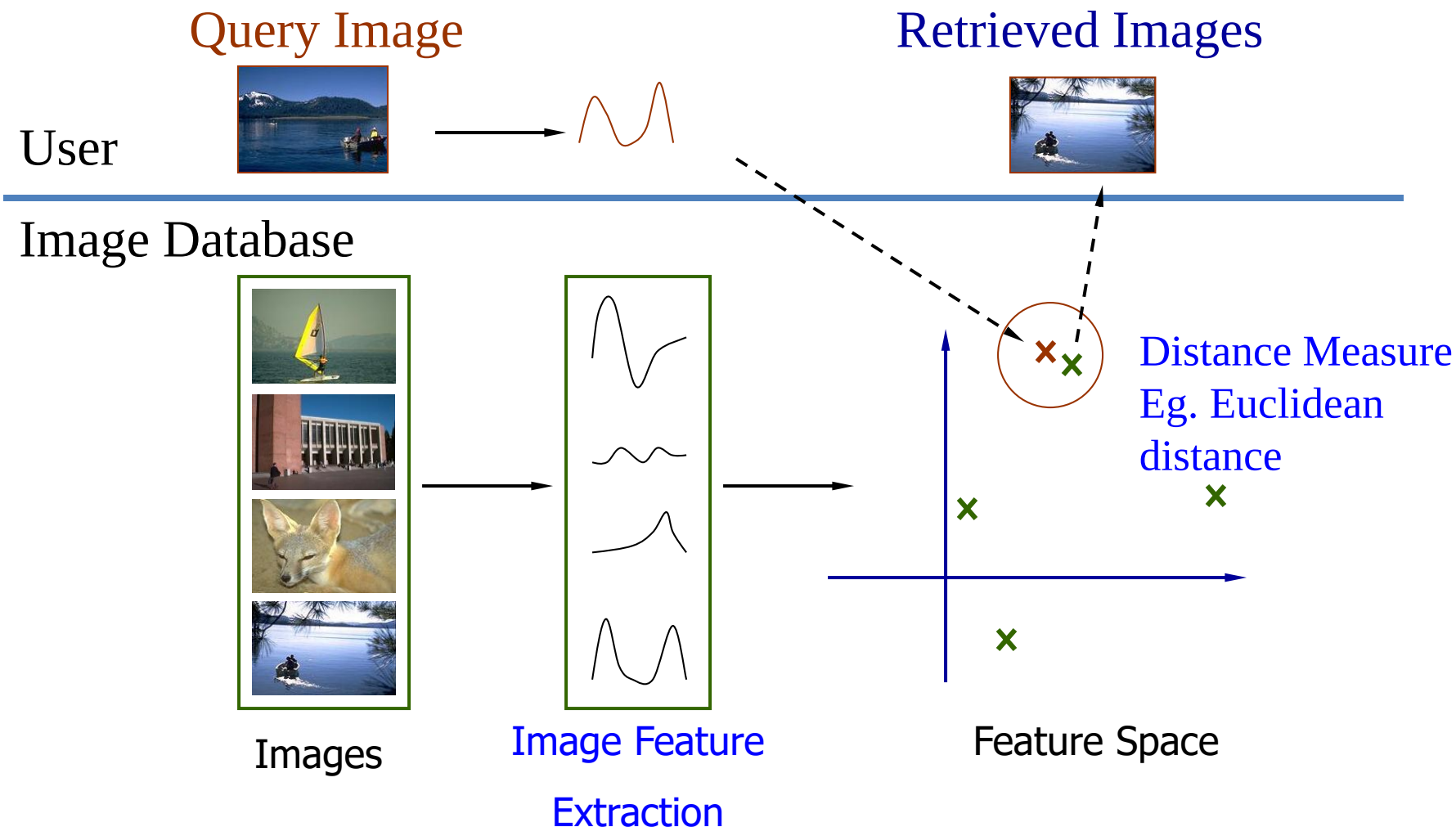
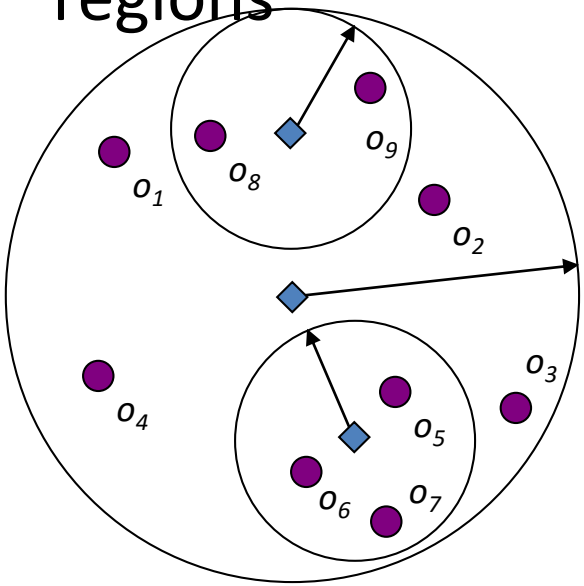


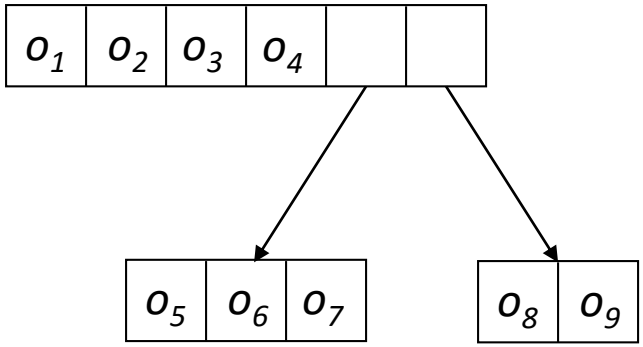
Image Features / Distance Measures



Tree based index organization

- Hierarchical decomposition of the space
- Using ball regions
  - Root node organizes some objects and some regions.
  - Internal nodes also organize other object and regions



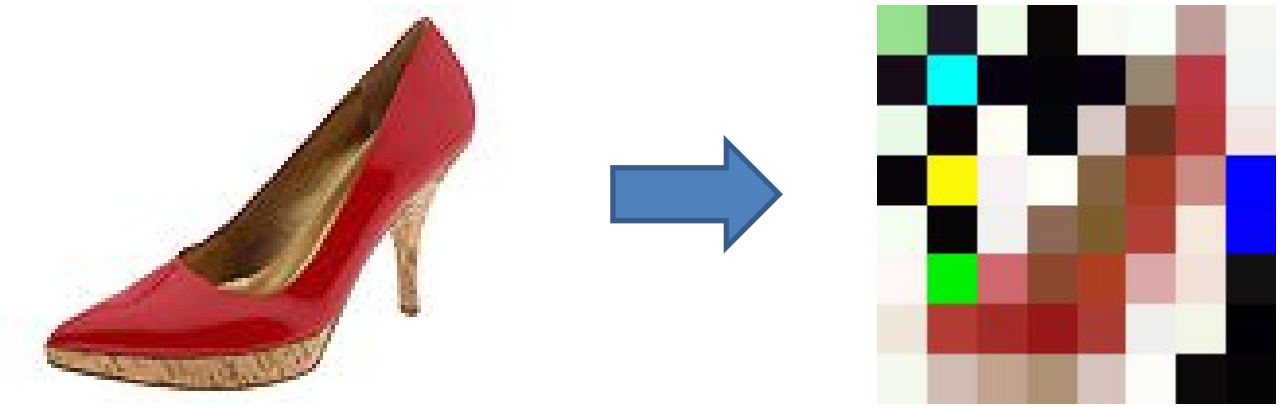


Approximate similarity search

- Approximate (or inexact) similarity search over-comes problems of exact similarity search when using traditional access methods.
  - Moderate improvement of performance with respect to the sequential scan.
  - Dimensionality curse

Approximate similarity search

- This is reasonable in many applications because the **feature-extraction** or the metric-space modelizations already involve an **approximation to reality**, and therefore a second approximation is usually acceptable (Chavez et al).

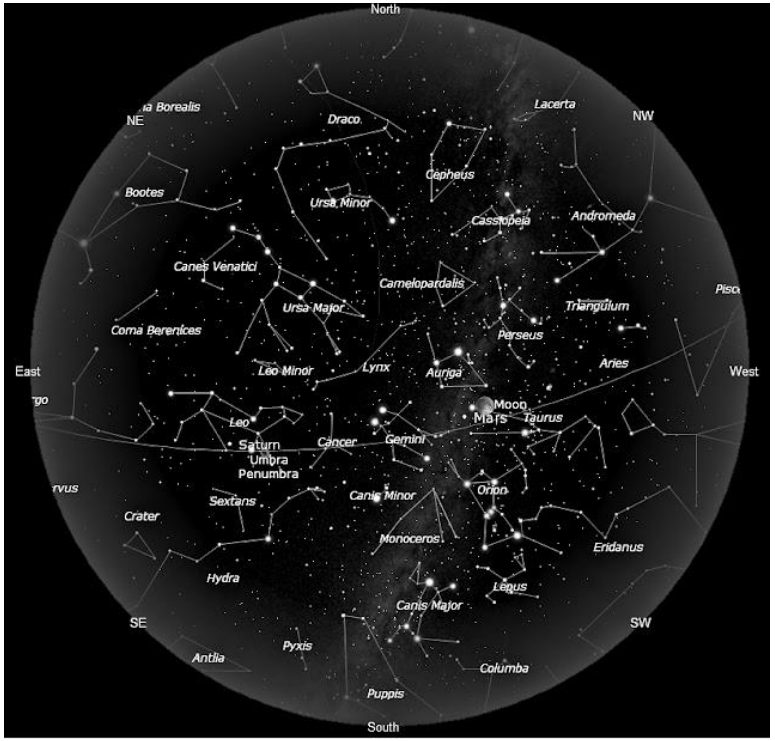


Permutation based indexes

Objects that are close to each other see the space in a "similar" way

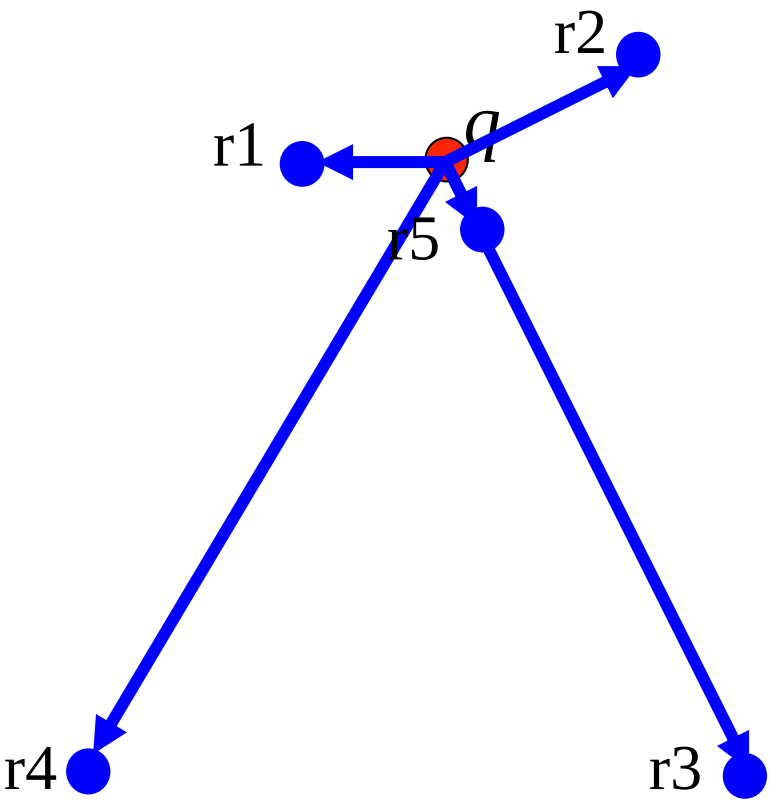
Suppose we chose a set of reference objects RO

Permutations of RO ordered according to the distances from two similar data objects are similar as well



- We can represent every data object o as an ordering of RO according to the distance from o
- We can measure the similarity between two data objects by measuring the similarity between the corresponding orderings

Permutation based transformation

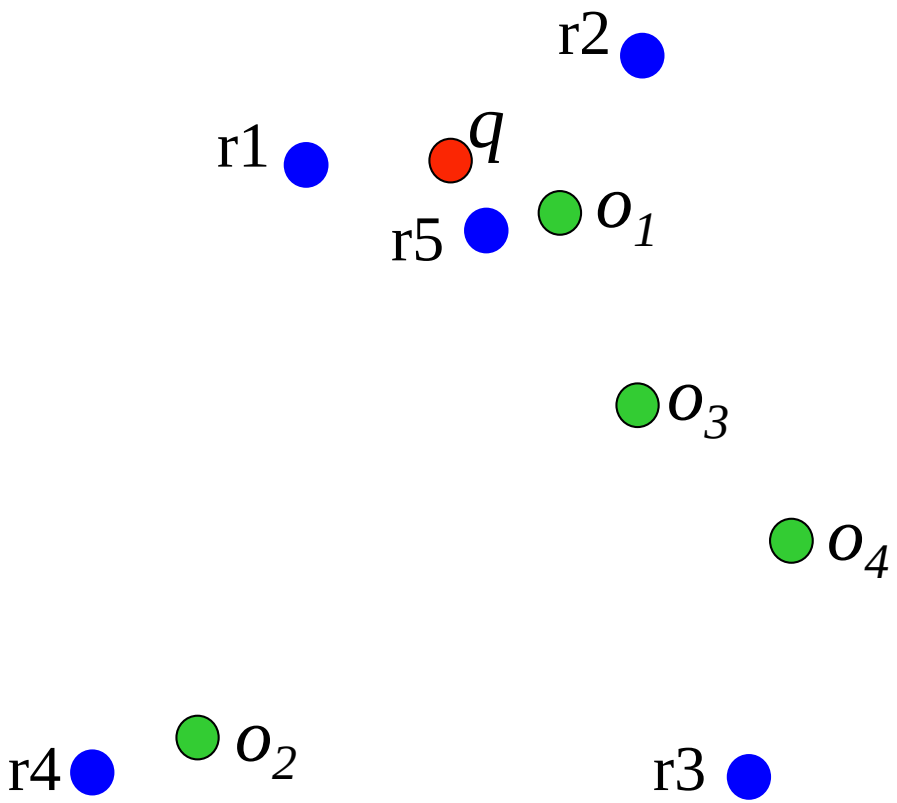


- Mapped objects:

r1 r2 r3 r4 r5

- p(q)= (2 3 4 5 1)

Permutation based transformation



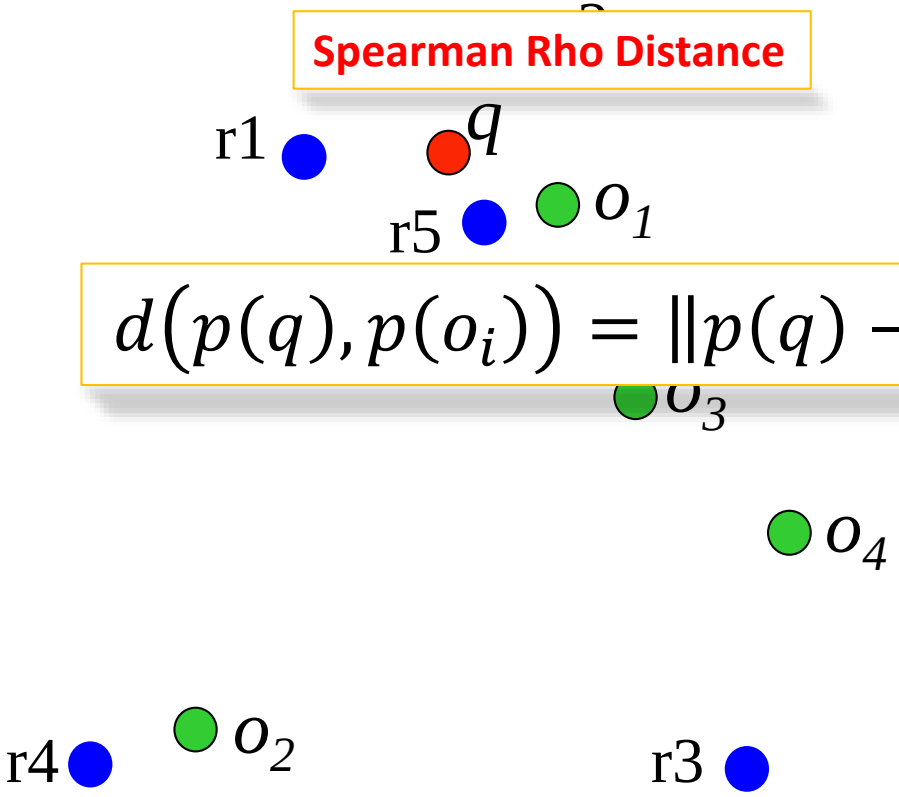
- Mapped objects:

r1 r2 r3 r4 r5

- p(q)= (2,3,4,5,1)

- $p(o_1)$= (3,2,4,5,1)
- $p(o_3)$= (4,2,3,5,1)
- $p(o_4)$= (4,3,1,5,2)
- $p(o_2)$= (4,5,2,1,3)

Permutation based transformation



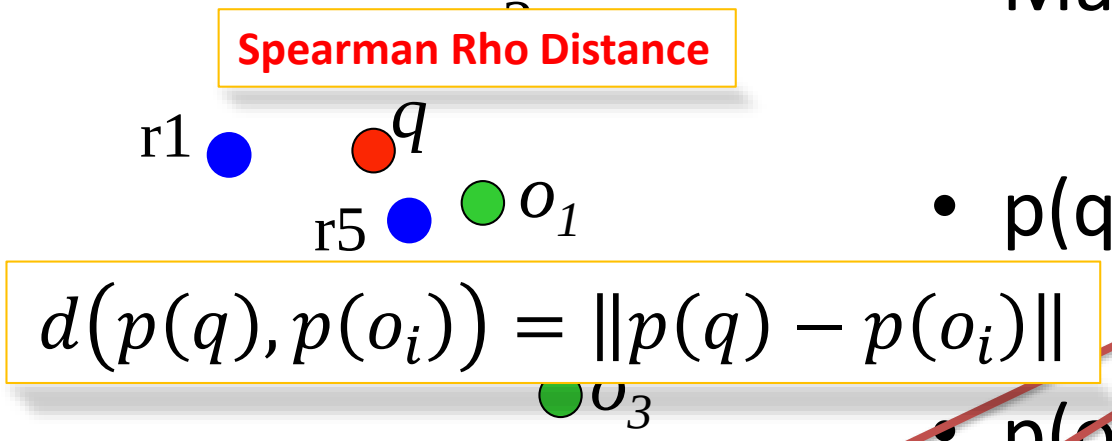
- Mapped objects:

	r1	r2	r3	r4	r5
• p(q)=	2	3	4	5	1

matching

• $p(o_1)$=	3	2	4	5	1
• $p(o_3)$=	4	2	3	5	1
• $p(o_4)$=	4	3	1	5	2
• $p(o_2)$=	4	5	2	1	3

Permutation based transformation



$$d\big(p(q), p(o_i)\big) = \|p(q) - p(o_i)\|$$

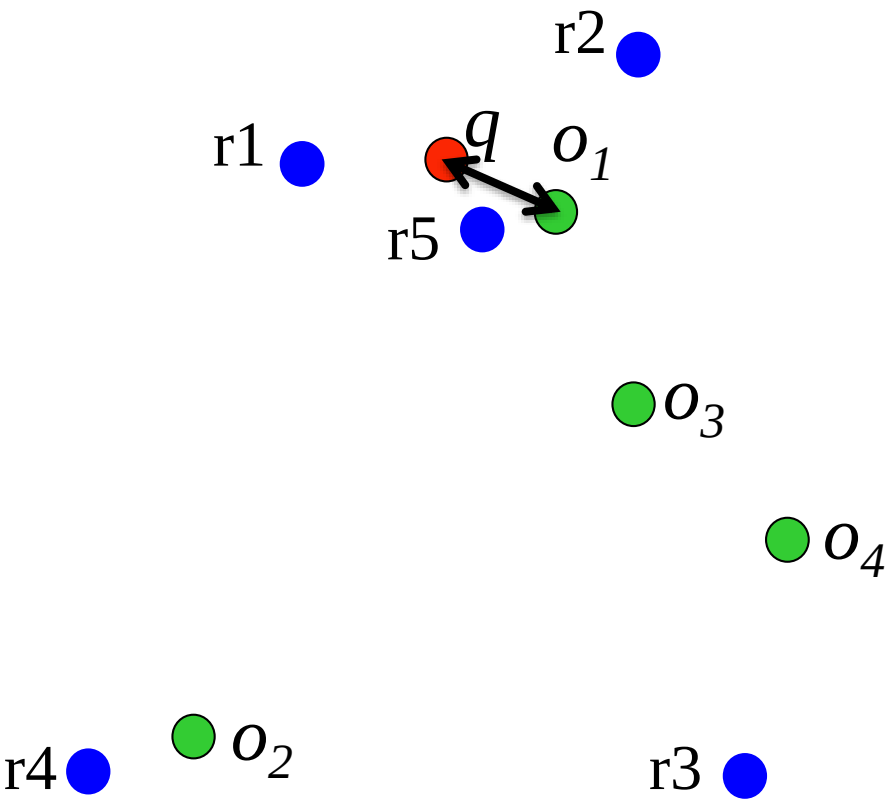
- Mapped objects:

r1 r2 r3 r4 r5

- p(q)= (2,3,4,5,1)
- $p(o_1)$= (3,2,4,5,1)
- $p(o_2)$= (4,5,2,1,3)
- $p(o_3)$= (4,2,3,5,1)

$$d_\rho(q, O_1) = \sqrt{(2-3)^2 + (3-2)^2 + (4-4)^2 + (5-5)^2 + (1-1)^2} = \sqrt{2}$$

Permutation based transformation



- Mapped objects:

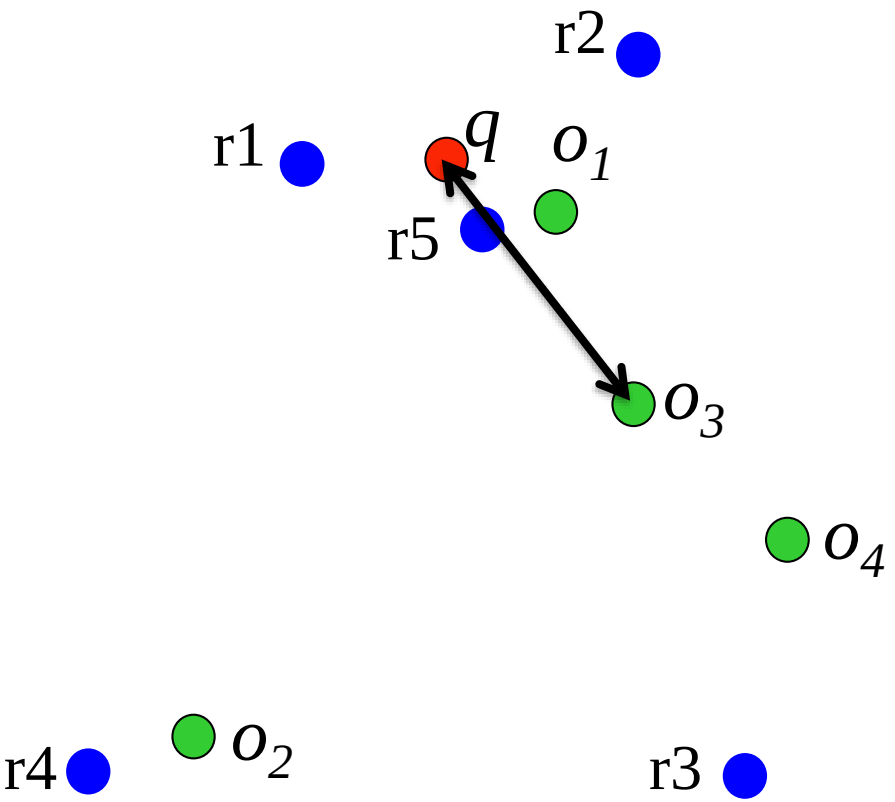
r1 r2 r3 r4 r5

- p(q)= (2,3,4,5,1)

d_ρ

- p(o_1)= (3,2,4,5,1) $\sqrt{2}$

Permutation based transformation



- Mapped objects:

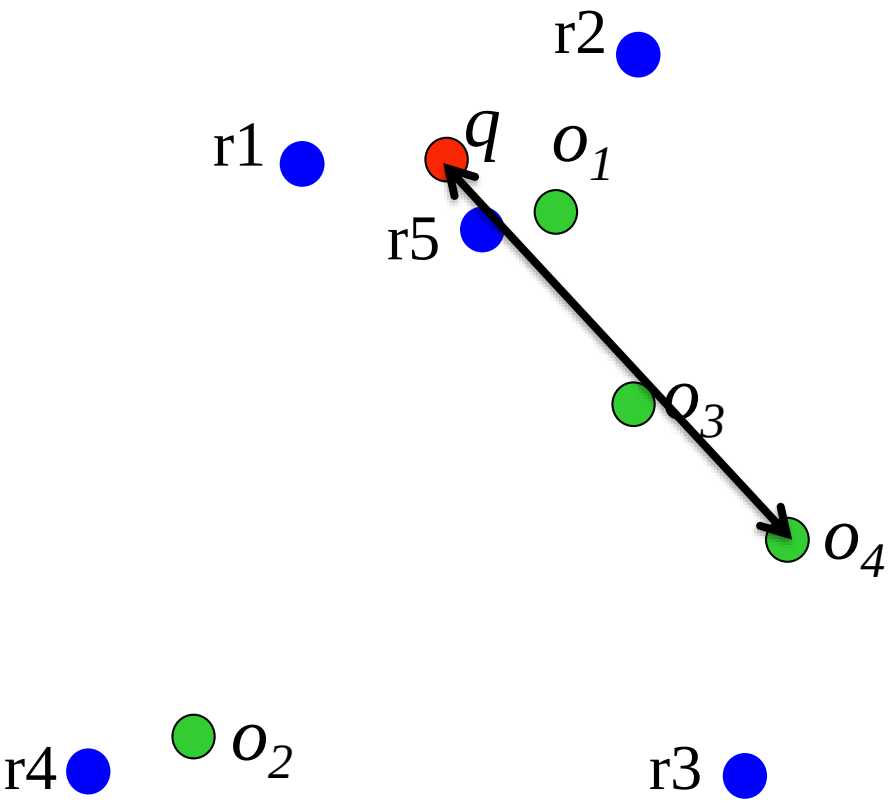
r1 r2 r3 r4 r5

- p(q)= (2,3,4,5,1)

d_ρ

- p(o_1)= (3,2,4,5,1) $\sqrt{2}$
- p(o_3)= (4,2,3,5,1) $\sqrt{6}$

Permutation based transformation

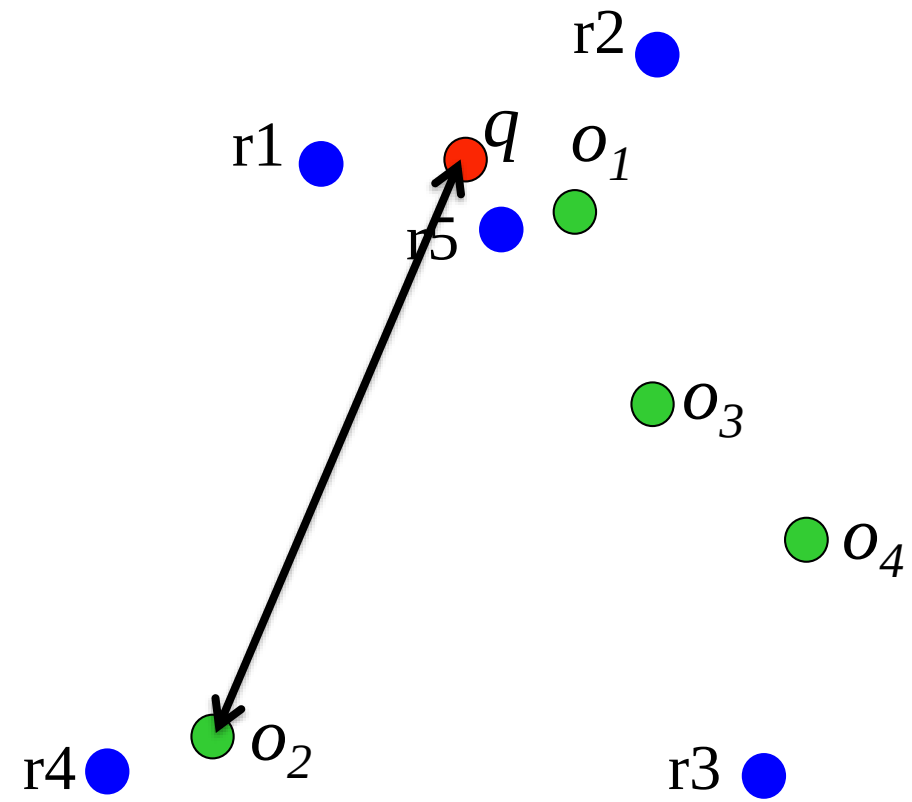


- Mapped objects:

r1 r2 r3 r4 r5 d_ρ

- p(q)= (2,3,4,5,1)
- p(o_1)= (3,2,4,5,1) $\sqrt{2}$
- p(o_3)= (4,2,3,5,1) $\sqrt{6}$
- p(o_4)= (4,3,1,5,2) $\sqrt{14}$

Permutation based transformation



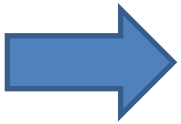
- Mapped objects:

	r1 r2 r3 r4 r5	d_ρ
• p(q)=	(2,3,4,5,1)	
• p(o_1)=	(3,2,4,5,1)	$\sqrt{2}$
• p(o_3)=	(4,2,3,5,1)	$\sqrt{6}$
• p(o_4)=	(4,3,1,5,2)	$\sqrt{14}$
• p(o_2)=	(4,5,2,1,3)	$\sqrt{28}$

Inverted File

- This information can be straightforwardly stored in **inverted file**, where **reference objects** represents **dictionary's** words and objects are documents (**posting lists**):

	r1	r2	r3	r4	r5
O_1 =	3	2	4	5	1
O_2 =	4	5	2	1	3
O_3 =	4	2	3	5	1
O_4 =	4	3	1	5	2



q		O_1	O_2	O_3	O_4
2	r_1 ->	3	4	4	4
3	r_2 ->	2	5	2	3
4	r_3 ->	4	2	3	1
5	r_4 ->	5	1	5	5
1	r_4 ->	1	3	1	2

Exploiting standard SE: How?

- We want to use a standard full text search engines (such as **Lucene**)
- And retrieve the documents by means of the standard **cosine similarity**,
- we are lazy, and don't want change a line of code of Lucene!
- *Complementing the Ranks*

Complementing rank orders

m = (6,6,6,6,6)

- Mapped objects:

r1 r2 r3 r4 r5

- p(q)= (2,3,4,5,1) p(q) = **m** - p(q) = (4,3,2,1,5)

- $p(o_1)$=(3,2,4,5,1) $p(o_1)$ = **m** - $p(o_1)$=(3,4,2,1,5)
- $p(o_3)$=(4,2,3,5,1) $p(o_3)$ = **m** - $p(o_3)$=(2,4,3,1,5)
- $p(o_4)$=(4,3,1,5,2) $p(o_4)$ = **m** - $p(o_4)$=(2,3,5,1,4)
- $p(o_2)$=(4,5,2,1,3) $p(o_2)$ = **m** - $p(o_2)$=(2,1,4,5,3)

Why?

- Instead of using the Euclidean distance, we use now the dot product
- distance vs similarity: we interested to ranking results rather than absolute values of the scores.

$$d\big(p(q), p(o_i)\big) = \|p(q) - p(o_i)\|$$

$$sim\big(p(q), p(o_i)\big) = p(q) * p(o_i)$$

$$p(q) * p(o_i) = \alpha - \beta\|p(q) - p(o_i)\|^2$$

Why?

- Instead of using the Euclidean distance, we use now the dot product
- distance vs similarity: we interested to ranking results rather than absolute values of the scores.

$$d\big(p(q), p(o_i)\big) = \|p(q) - p(o_i)\|$$

$$sim\big(p(q), p(o_i)\big) = p(q) * p(o_i)$$

const

$$p(q) * p(o_i) = \alpha - \beta\|p(q) - p(o_i)\|^2$$

Permutation based transformation

p(q) = (4,3,2,1,5)

p(o_1) = (3,4,2,1,5) 54

p(o_3) = (2,4,3,1,5) 52

p(o_4) = (2,3,5,1,4) 48

p(o_2) = (2,1,4,5,3) 39

*

- Mapped objects:

r1 r2 r3 r4 r5

- p(q)= (2,3,4,5,1) d

- p(o_1)= (3,2,4,5,1) $\sqrt{2}$
- p(o_3)= (4,2,3,5,1) $\sqrt{6}$
- p(o_4)= (4,3,1,5,2) $\sqrt{14}$
- p(o_2)= (4,5,2,1,3) $\sqrt{28}$

Surrogate Text Representation

- Assign a keyword to each Reference Object and build a dictionary

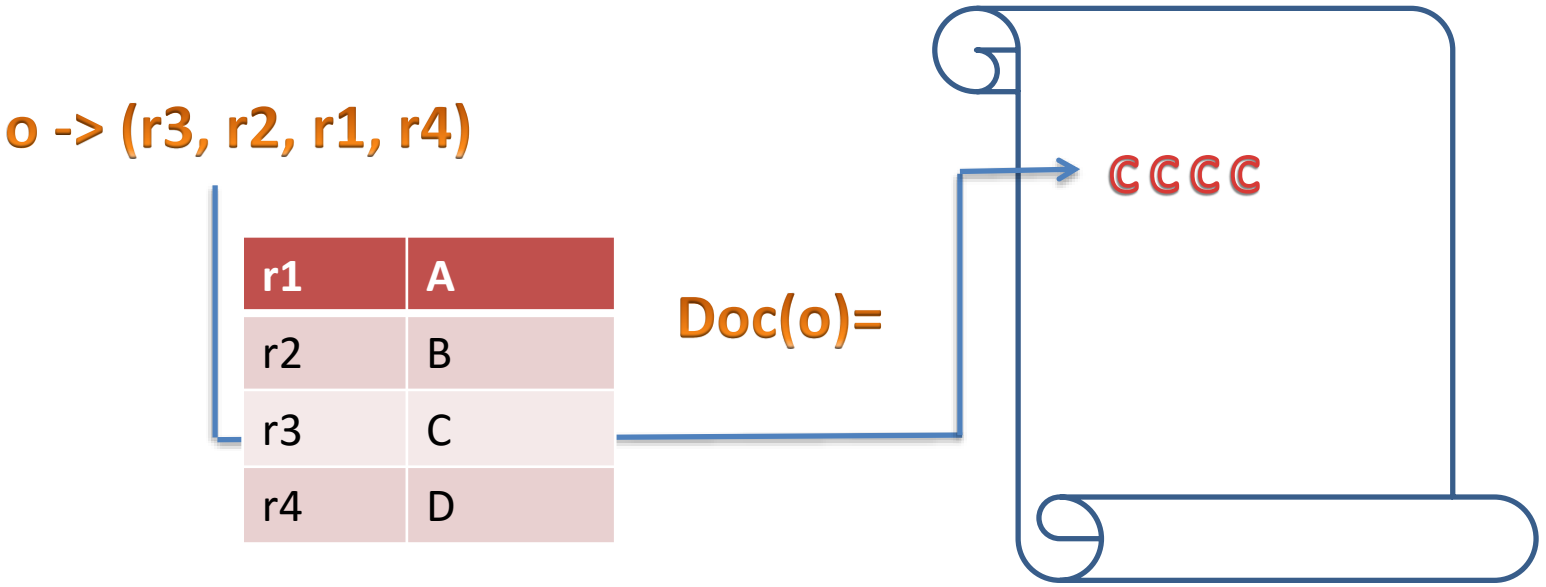
Rerefence obj	keyword
r1	A
r2	B
r3	C
r4	D

Surrogate Text Representation

- For each object we create a textual document by repeating the reference object-keyword on the basis of its ranking, e.g.:

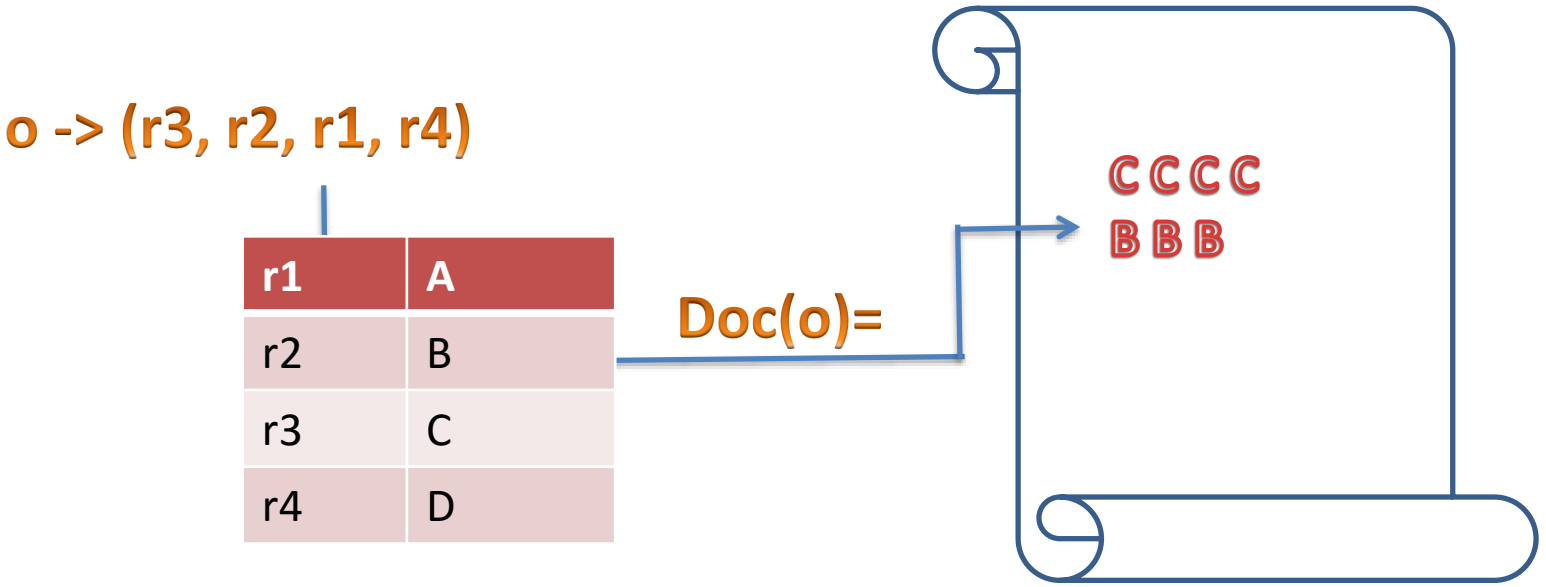
Surrogate Text Representation

- For each object we create a textual document by repeating the reference object-keyword on the basis of its ranking, e.g.:



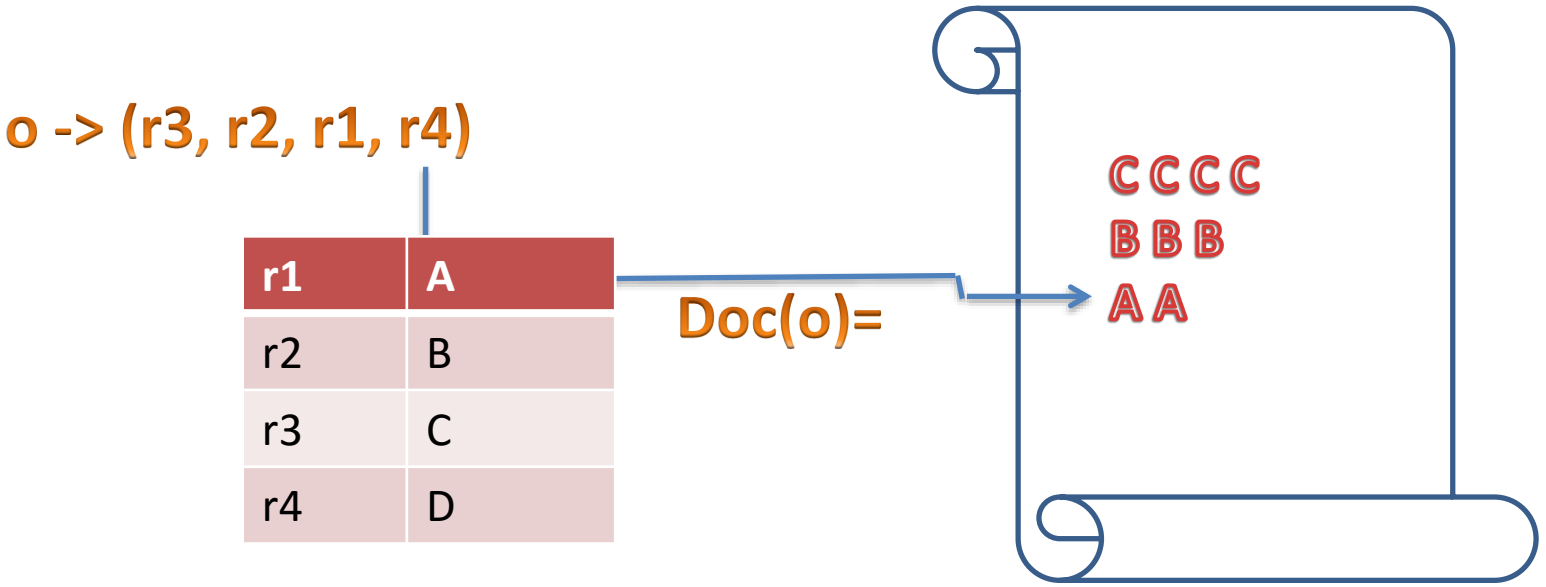
Surrogate Text Representation

- For each object we create a textual document by repeating the reference object-keyword on the basis of its ranking, e.g.:



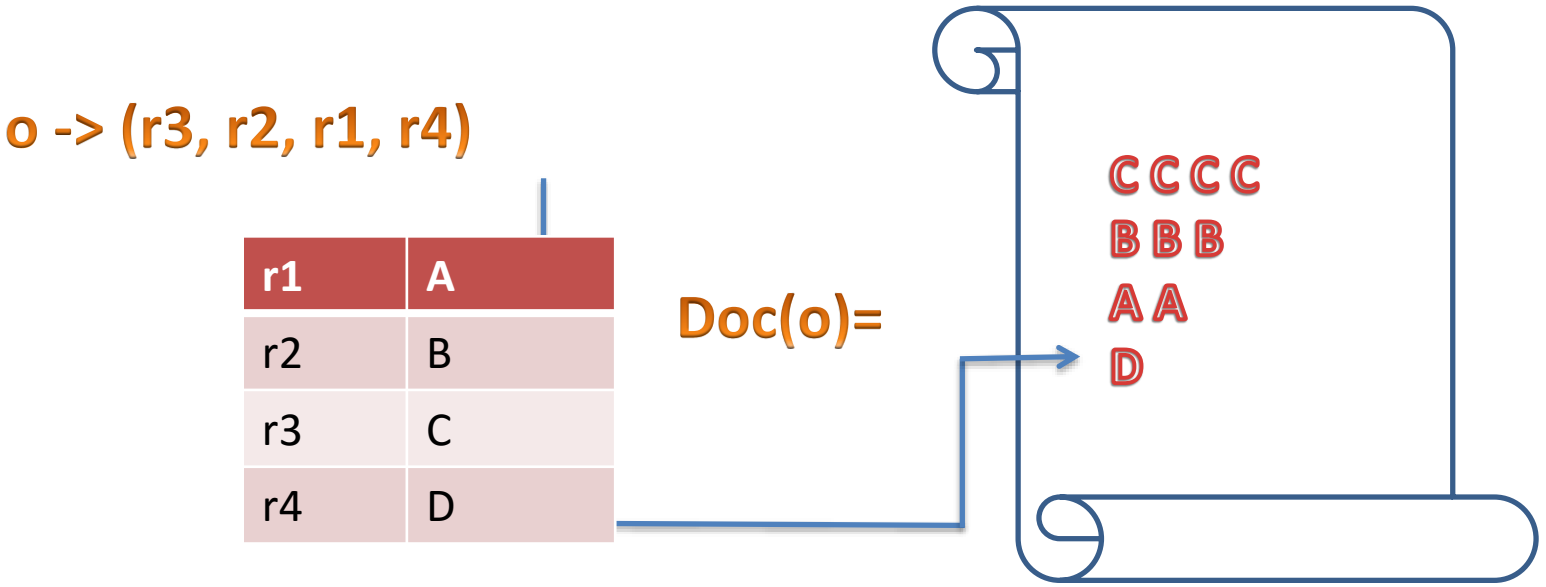
Surrogate Text Representation

- For each object we create a textual document by repeating the reference object-keyword on the basis of its ranking, e.g.:



Surrogate Text Representation

- For each object we create a textual document by repeating the reference object-keyword on the basis of its ranking, e.g.:



Surrogate Text Representation

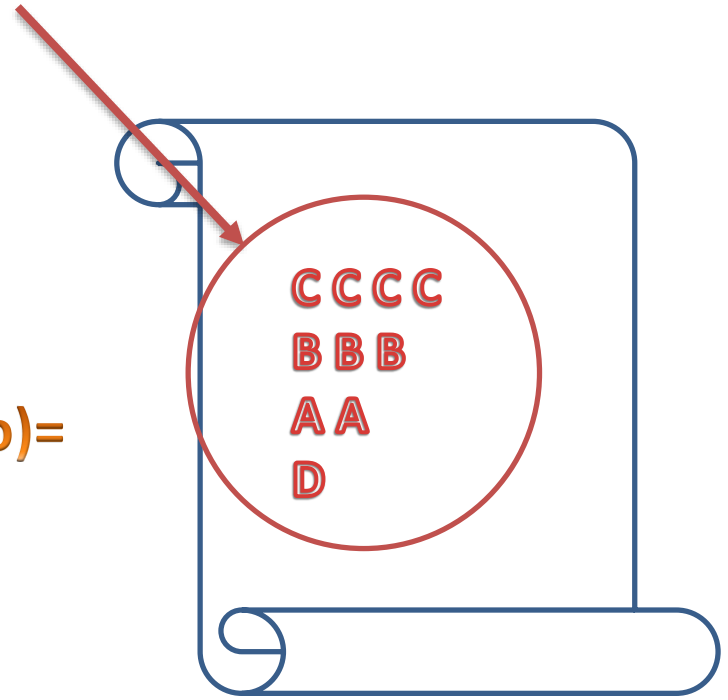
- We call this text:

Surrogate Text Representation (SRT)

o -> (r3, r2, r1, r4)

r1	A
r2	B
r3	C
r4	D

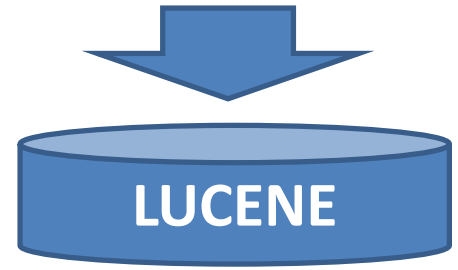
Doc(o)=



Surrogate Text Representation

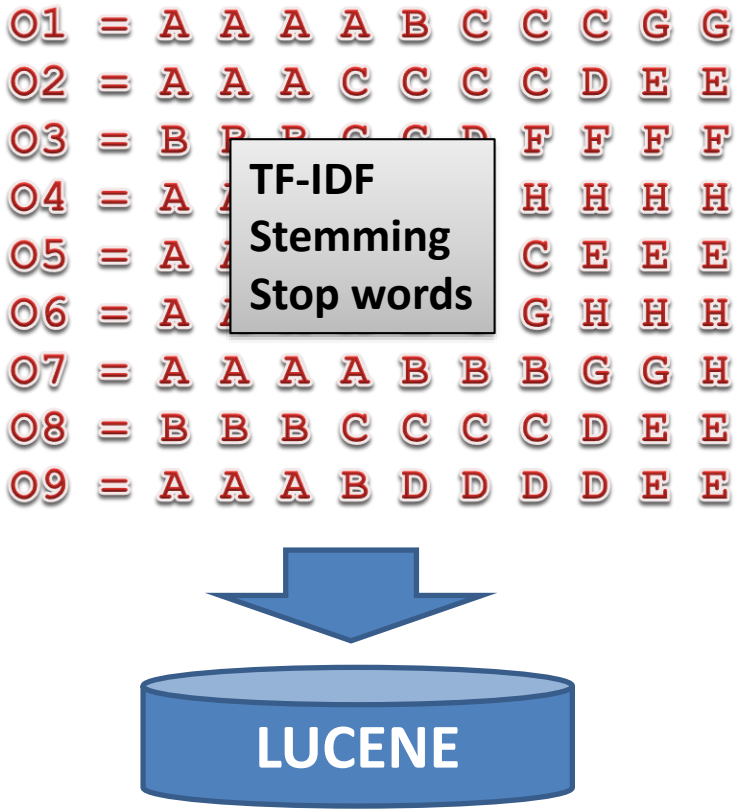
- Index these documents and query by using the same transformation for queries

```
O1 = A A A A B C C C G G
O2 = A A A C C C C D E E
O3 = B B B C C D F F F F
O4 = A A A D D E H H H H
O5 = A A B B B B C E E E
O6 = A A A A E G G H H H
O7 = A A A A B B B G G H
O8 = B B B C C C C D E E
O9 = A A A B D D D D E E
```



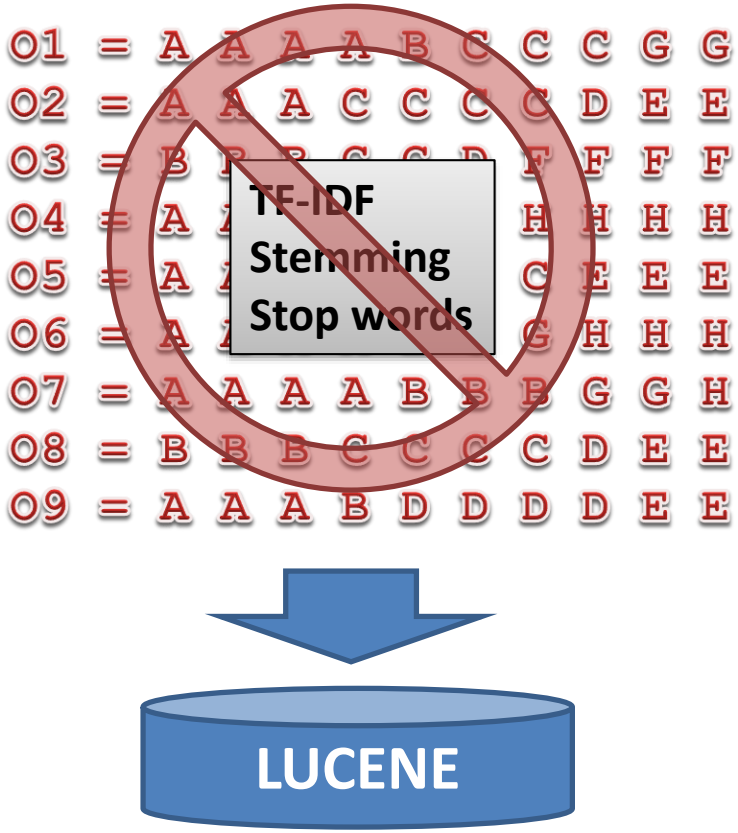
Surrogate Text Representation

- Index these documents and query by using the same transformation for queries



Surrogate Text Representation

- Index these documents and query by using the same transformation for queries



Optmizing in space the inverted file

- Usually a full text **inverted file is sparse matrix**, we there are lot of 0s (big dictionary).
- In our case we would obtain a small dictionary and each document would contain all their terms,
- for instance with 20,000 reference objects (terms) and 1 million documents the size of the inverted file is 20 billions items!

Optimizing in space the inverted file

- In order to reduce the size of the inverted index we can exploit the idea of taking just **k_x closest (m) reference objects** to represent any object that has to be indexed.
- $k_x \ll m$
- In practice we must compare partial ranking

Optimizing in space the inverted file (cont)

- Example *m=8*

	o1	o2	o3	o4	o5	o6	o7	o8	o9
r1	1	2	5	2	3	1	1	5	2
r2	4	6	2	5	1	6	2	2	4
r3	2	1	3	6	4	8	6	1	8
r4	5	4	4	3	5	5	5	4	1
r5	7	3	6	4	2	4	7	3	3
r6	6	5	1	7	6	7	8	7	7
r7	3	7	8	8	7	3	3	8	5
r8	8	8	7	1	8	2	4	6	6

Optimizing in space the inverted file (cont)

- Example $m=8$, $k_x = 4$

	o1	o2	o3	o4	o5	o6	o7	o8	o9
r1	**1**	**2**	5	**2**	**3**	**1**	**1**	5	**2**
r2	**4**	6	**2**	5	**1**	6	**2**	**2**	**4**
r3	**2**	**1**	**3**	6	**4**	8	6	**1**	8
r4	5	**4**	**4**	**3**	5	5	5	**4**	**1**
r5	7	**3**	6	**4**	**2**	**4**	7	**3**	**3**
r6	6	5	**1**	7	6	7	8	7	7
r7	**3**	7	8	8	7	**3**	**3**	8	5
r8	8	8	7	**1**	8	**2**	**4**	6	6

Optimizing in space the inverted file (cont)

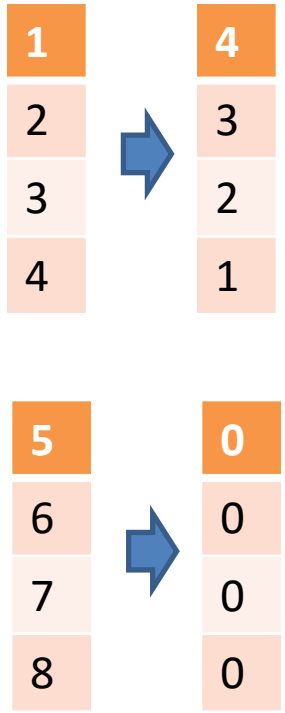
- Example *m=8,* k_x *= 4, if value > 4 put 5*

	o1	o2	o3	o4	o5	o6	o7	o8	o9
r1	1	2	5	2	3	1	1	5	2
r2	4	5	2	5	1	5	2	2	4
r3	2	1	3	5	4	5	5	1	5
r4	5	4	4	3	5	5	5	4	1
r5	5	3	5	4	2	4	5	3	3
r6	5	5	1	5	5	5	5	5	5
r7	3	5	5	5	5	3	3	5	5
r8	5	5	5	1	5	2	4	5	5

Optimizing in space the inverted file (cont)

- Example $m=8$, $k_x = 4$
- **complement the ranks** into the inverted file $k_x + 1 - o_i$

	o1	o2	o3	o4	o5	o6	o7	o8	o9
r1	4	3	0	3	2	4	4	0	3
r2	1	0	3	0	4	0	3	3	1
r3	3	4	2	0	1	0	0	4	0
r4	0	1	1	2	0	0	0	1	4
r5	0	2	0	1	3	1	0	2	2
r6	0	0	4	0	0	0	0	0	0
r7	2	0	0	0	0	2	2	0	0
r8	0	0	0	4	0	3	1	0	0



Generating the inverted file

- Very trivial: we define an arbitrary the **dictionary** of the reference object, for instance:

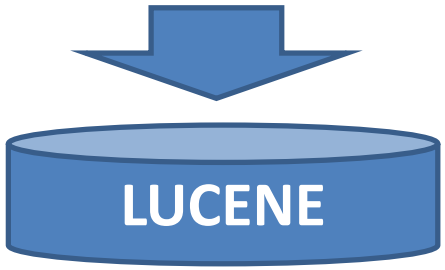
r1	A
r2	B
r3	C
r4	D
r5	E
r6	F
r7	G
r8	H

Generating the inverted file (cont)

- And generate a **textual representation** of the object by simply repeating the terms associated with the reference objects

	o1	o2	o3	o4	o5	o6	o7	o8	o9
A	4	3	0	3	2	4	4	0	3
B	1	0	3	0	4	0	3	3	1
C	3	4	2	0	1	0	0	4	0
D	0	1	1	2	0	0	0	1	4
E	0	2	0	1	3	1	0	2	2
F	0	0	4	0	0	0	0	0	0
G	2	0	0	0	0	2	2	0	0
H	0	0	0	4	0	3	1	0	0

```
o1 = A A A A B C C C G G
o2 = A A A C C C C D E E
o3 = B B B C C D F F F F
o4 = A A A D D E H H H H
o5 = A A B B B B C E E E
o6 = A A A A E G G H H H
o7 = A A A A B B B G G H
o8 = B B B C C C C D E E
o9 = A A A B D D D D E E
```



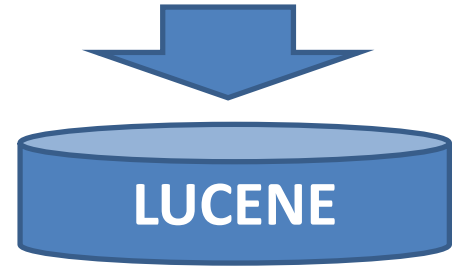
Opmizing in space the inverted file

- The suggested minimum number of reference object is about $2\sqrt{N}$

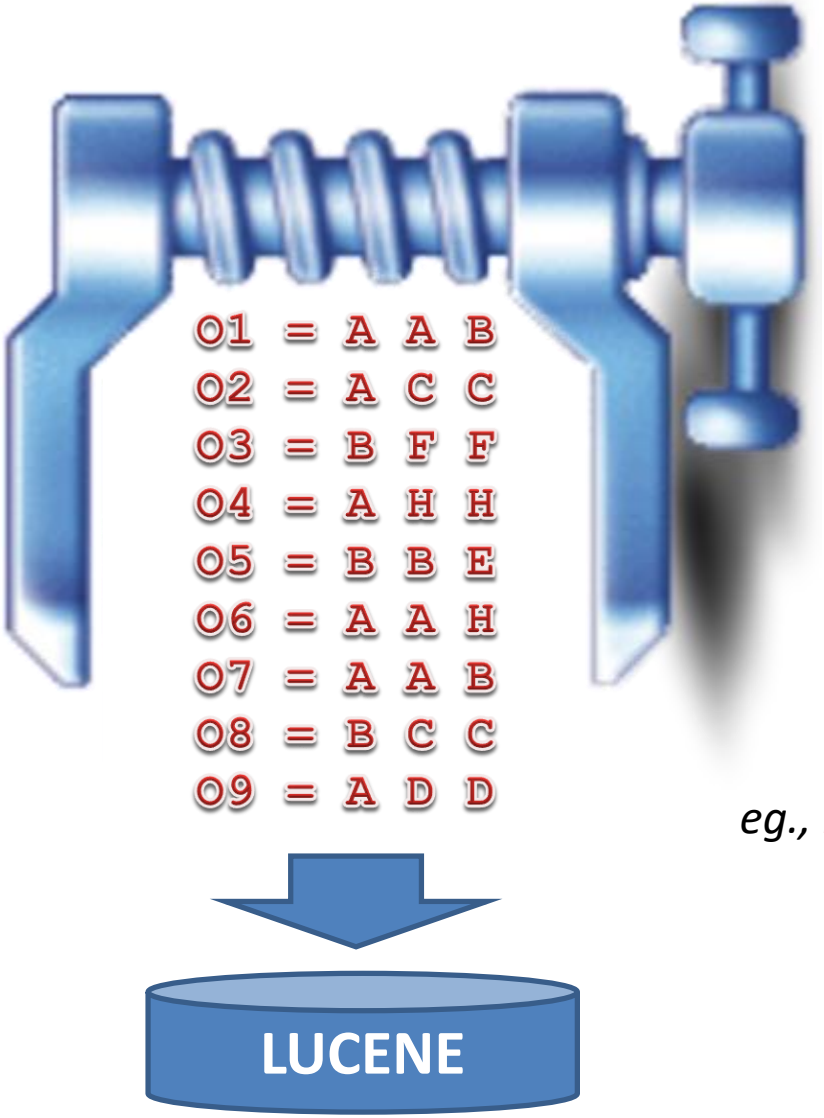
- For instance with N=100 millions images the number of reference object should be 20 thousands reference objects (terms)
- the size of the inverted file is 2 T items.

Dataset using all references

01 = A A A A B C C C G G
02 = A A A C C C C D E E
03 = B B B C C D F F F F
04 = A A A D D E H H H H
05 = A A B B B B C E E E
06 = A A A A E G G H H H
07 = A A A A B B B G G H
08 = B B B C C C C D E E
09 = A A A B D D D D E E



Dataset using only top 2 references



eg., k_i=50 instead of 20,000

Why it works?

- Text search engine use a normalized version of the scalar product called cosine similarity for comparing vectors of term frequencies. If we neglect the normalization:

$$sim\big(p(q), p(o)\big) = p(q) * p(o)$$

- it is possible to prove that we obtain the same results if we calculated the rho-distance of the reference object ranks.

Why it works?

- Rank vector $p(o)$= (3, 1, 2)

- Complemented rank vector:
  $p(o)$= (4, 4, 4) – $p(o)$
  $p(o)$= (1, 3, 2)

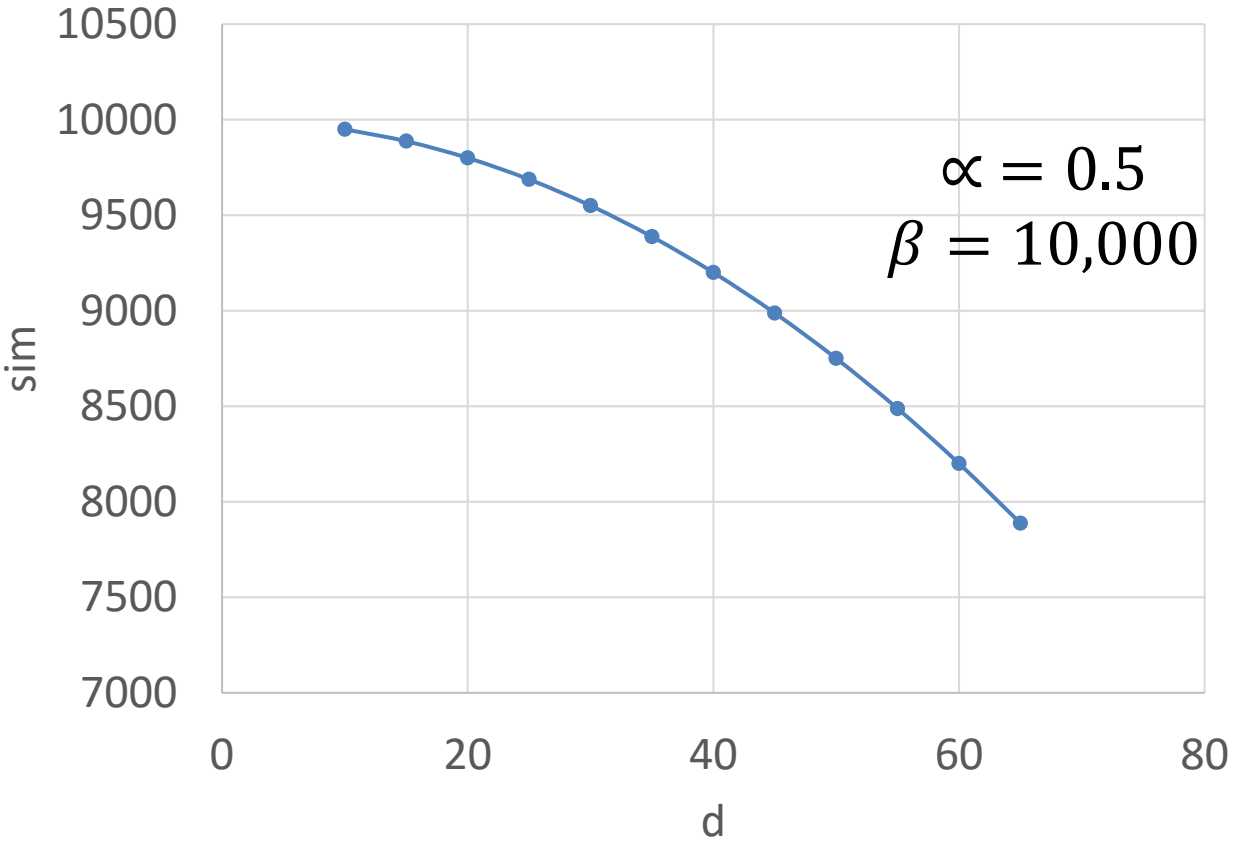
- we do the same for queries
  $p(q) \rightarrow p(q)$

Why it works?

$$sim\big(p(q), p(o)\big) = p(q) * p(o)$$
$$= \alpha - \beta \|p(q) - p(o)\|^2$$

- Scores is a monotonic transformation of the rho distance between permutated vectors
- However, we interested to ranking results rather than absolute values of the scores.

Monotonic transformation



Rigorous proof

- Standard Cosine similarity is:

$$sim\big(p(q), p(o)\big) = \frac{p(q) * p(o)}{\|p(q)\| \|p(o)\|}$$

- Complemented rank vectors:

$$p(o) = \boldsymbol{k_x} - p(o)$$
$$p(q) = \boldsymbol{k_q} - p(q)$$

- Where

$$\boldsymbol{k_x} = (k_x + 1, \ldots, k_x + 1)$$
$$\boldsymbol{k_q} = (k_q + 1, \ldots, k_q + 1)$$

	p(o)	p(q)
r1	1	2
r2	4	6
r3	2	1
r4	5	4
r5	7	3
r6	6	5
r7	3	7
r8	8	8

Rigorous proof

- Standard Cosine similarity is:

$$sim\big(p(q), p(o)\big) = \frac{p(q) * p(o)}{\|p(q)\| \|p(o)\|}$$

- Complemented rank vectors:

$$p(o) = \boldsymbol{k_x} - p(o)$$
$$p(q) = \boldsymbol{k_q} - p(q)$$

- Where

$$\boldsymbol{k_x} = (k_x + 1, \ldots, k_x + 1)$$
$$\boldsymbol{k_q} = (k_q + 1, \ldots, k_q + 1)$$

k_x= 4 k_q= 3

	p(o)	p(q)
r1	**1**	**2**
r2	**4**	6
r3	**2**	**1**
r4	5	**4**
r5	7	**3**
r6	6	5
r7	**3**	7
r8	8	8

Rigorous proof

- Standard Cosine similarity is:

$$sim\big(p(q), p(o)\big) = \frac{p(q) * p(o)}{\|p(q)\| \|p(o)\|}$$

- Complemented rank vectors:

$$p(o) = \boldsymbol{k_x} - p(o)$$
$$p(q) = \boldsymbol{k_q} - p(q)$$

- Where

$$\boldsymbol{k_x} = (k_x + 1, \ldots, k_x + 1)$$
$$\boldsymbol{k_q} = (k_q + 1, \ldots, k_q + 1)$$

k_x= 4 k_q= 3

	p(o)	p(q)
r1	1	2
r2	4	4
r3	2	1
r4	5	4
r5	5	3
r6	5	4
r7	3	4
r8	5	4

Rigorous proof

- Standard Cosine similarity is:

$$sim(p(q), p(o)) = \frac{p(q) * p(o)}{\|p(q)\| \|p(o)\|}$$

$k_x = 4$ $k_q = 3$

	p(o)	p(q)
r1	4	2
r2	1	0
r3	3	3
r4	0	0
r5	0	1
r6	0	0
r7	2	0
r8	0	0

$4+1-o$ $3+1-q$

Rigorous proof

- Standard Cosine similarity is:

$$sim(p(q), p(o)) = \frac{p(q) * p(o)}{\|p(q)\| \|p(o)\|}$$

$k_x = 4$ $k_q = 3$

	p(o)	p(q)
r1	4	2
r2	1	0
r3	3	3
r4	0	0
r5	0	1
r6	0	0
r7	2	0
r8	0	0

$p(q)$ is a permutation of the vector $(1,2,3, \dots, k, 0, \dots, 0)$

Therefore $\|p(o)\| = \sqrt{\frac{k(k+1)(2k+1)}{6}} = const$

Rigorous proof

- Standard Cosine similarity is:

$$sim(p(q), p(o)) = \frac{p(q) * p(o)}{\|p(q)\| \|p(o)\|}$$

$k_x = 4$ $k_q = 3$

	p(o)	p(q)
r1	4	2
r2	1	0
r3	3	3
r4	0	0
r5	0	1
r6	0	0
r7	2	0
r8	0	0

$p(q)$ is a permutation of the vector $(1,2,3, \dots, k, 0, \dots, 0)$

Therefore $\|p(o)\| = \sqrt{\frac{k(k+1)(2k+1)}{6}} = const$

Rigorous proof

$$sim\big(p(q), p(o)\big) \propto p(q) * p(o) =$$

$$\big(\boldsymbol{k_q} - p(q)\big) * \big(\boldsymbol{k_x} - p(o)\big) =$$

$$\boldsymbol{k_q}*\boldsymbol{k_x} - \boldsymbol{k_q}*p(o) - \boldsymbol{k_x}* p(q) + p(q) * p(o)$$

k_x= 4 k_q= 3

	x	y
r1	**4**	**2**
r2	**1**	0
r3	**3**	**3**
r4	0	0
r5	0	**1**
r6	0	0
r7	**2**	0
r8	0	0

Rigorous proof

$$sim\big(p(q), p(o)\big) \propto p(q) * p(o) =$$

$$\big(\boldsymbol{k_q} - p(q)\big) * \big(\boldsymbol{k_x} - p(o)\big) =$$

$$\boldsymbol{k_q}{*}\boldsymbol{k_x} - \boldsymbol{k_q}{*}p(o) - \boldsymbol{k_x}{*}\, p(q) + p(q) * p(o)$$

$$(4,4,4,4,4,4,4,4) * (5,5,5,5,5,5,5,5) = 4 * 5 + \cdots 4 * 5 = 160$$

k_x= 4 k_q= 3

	x	y
r1	4	2
r2	1	0
r3	3	3
r4	0	0
r5	0	1
r6	0	0
r7	2	0
r8	0	0

Rigorous proof

$$sim\big(p(q), p(o)\big) \propto p(q) * p(o) =$$

$$\big(\boldsymbol{k_q} - p(q)\big) * \big(\boldsymbol{k_x} - p(o)\big) =$$

$$\boldsymbol{k_q}*\boldsymbol{k_x} - \boldsymbol{k_q}*p(o) - \boldsymbol{k_x}*\, p(q) + p(q) * p(o)$$

$$(4,4,4,4,4,4,4,4) * (1,4,2,5,5,5,3,5\ ) =$$
$$4 * 1 + 4 * 4 + 4 * 2 + 4 * 5 + 4 * 5 + \cdots =$$
$$4 * (1 + 4 + 2 + 5 + 5 + 5 + 3 + 5) =$$
$$4 * (1 + 2 + 3 + 4 + 5 + 5 + 5 + 5) =$$
$$120$$

k_x= 4 k_q= 3

	x	y
r1	4	2
r2	1	0
r3	3	3
r4	0	0
r5	0	1
r6	0	0
r7	2	0
r8	0	0

Rigorous proof

$$sim\big(p(q), p(o)\big) \propto p(q) * p(o) =$$

$$\big(\boldsymbol{k_q} - p(q)\big) * \big(\boldsymbol{k_x} - p(o)\big) =$$

$$\boldsymbol{k_q}*\boldsymbol{k_x} - \boldsymbol{k_q}*p(o) - \boldsymbol{k_x}*p(q) + p(q) * p(o)$$

$$(5,5,5,5,5,5,5,5) * (2, 4, 1, 4, 3, 4, 4, 4) =$$
$$5 * 2 + 5 * 4 + 5 * 1 + 5 * 4 + 5 * 3 + \cdots =$$
$$5 * (2 + 4 + 1 + 4 + 3 + 4 + 4 + 4) =$$
$$5 * (1 + 2 + 3 + 4 + 4 + 4 + 4 + 4) =$$
$$130$$

k_x= 4 k_q= 3

	x	y
r1	4	2
r2	1	0
r3	3	3
r4	0	0
r5	0	1
r6	0	0
r7	2	0
r8	0	0

Rigorous proof

$$sim\big(p(q), p(o)\big) \propto= L\big(m, k_x, k_q\big) + p(q) * p(o)$$

k_x= 4 k_q= 3

	x	y
r1	**4**	**2**
r2	**1**	0
r3	**3**	**3**
r4	0	0
r5	0	**1**
r6	0	0
r7	**2**	0
r8	0	0

Rigorous proof

$$sim\big(p(q), p(o)\big) \propto= L\big(m, k_x, k_q\big) + p(q) * p(o)$$

- In general, the following relationship holds, between Euclidean distance and dot product

$$d(x,y)^2 = \|x - y\|^2 = \|x\|^2 + \|y\|^2 - 2\mathrm{x} * \mathrm{y}$$

$$\mathrm{x} * \mathrm{y} = \frac{\|x\|^2 + \|y\|^2 - d(x,y)^2}{2}$$

Rigorous proof

$$sim\big(p(q), p(o)\big) \propto= L\big(m, k_x, k_q\big) + p(q) * p(o)$$

$$sim\big(p(q), p(o)\big) \propto$$

$$L\big(m, k_x, k_q\big) + \frac{\|p(q)\|^2 + \|p(o)\|^2 - d\big(p(q), p(o)\big)^2}{2}$$

const

Rigorous proof

$$sim\big(p(q), p(o)\big) \propto= L\big(m, k_x, k_q\big) + p(q) * p(o)$$

$$sim\big(p(q), p(o)\big) \propto L'\big(m, k_x, k_q\big) - \frac{1}{2} d\big(p(q), p(o)\big)^2$$

The cosine similarity of two complemented rank vectors is a monotonic transformation of the rho distance between their corresponding rank vectors

An example

	o1	o2	o3	o4	o5	o6	o7	o8	o9
r1	1	2	5	2	3	1	1	5	2
r2	4	5	2	5	1	5	2	2	4
r3	2	1	3	5	4	5	5	1	5
r4	5	4	4	3	5	5	5	4	1
r5	5	3	5	4	2	4	5	3	3
r6	5	5	1	5	5	5	5	5	5
r7	3	5	5	5	5	3	3	5	5
r8	5	5	5	1	5	2	4	5	5

	q
r1	2
r2	4
r3	1
r4	4
r5	3
r6	4
r7	4
r8	4

An example

$$d\big(p(q), p(o)\big)^2$$

	o1	o2	o3	o4	o5	o6	o7	o8	o9
r1	1	2	5	2	3	1	1	5	2
r2	4	5	2	5	1	5	2	2	4
r3	2	1	3	5	4	5	5	1	5
r4	5	4	4	3	5	5	5	4	1
r5	5	3	5	4	2	4	5	3	3
r6	5	5	1	5	5	5	5	5	5
r7	3	5	5	5	5	3	3	5	5
r8	5	5	5	1	5	2	4	5	5
d^2	10	4	32	30	24	26	28	16	28

	q
r1	2
r2	4
r3	1
r4	4
r5	3
r6	4
r7	4
r8	4

An example

$$d\big(p(q), p(o)\big)^2$$

	o1	o2	o3	o4	o5	o6	o7	o8	o9		q
r1	1	2	5	2	3	1	1	5	2	r1	2
r2	4	5	2	5	1	5	2	2	4	r2	4
r3	2	1	3	5	4	5	5	1	5	r3	1
r4	5	4	4	3	5	5	5	4	1	r4	4
r5	5	3	5	4	2	4	5	3	3	r5	3
r6	5	5	1	5	5	5	5	5	5	r6	4
r7	3	5	5	5	5	3	3	5	5	r7	4
r8	5	5	5	1	5	2	4	5	5	r8	4
d^2	10	4	32	30	24	26	28	16	28		

order

2
1
8
5
6
7
9
4
3

- The result set is:
- o2, o1, o8, o5, o6, o7, o9, o4, o3

An example

$p(q) * p(o)$

	o1	o2	o3	o4	o5	o6	o7	o8	o9
r1	4	3	0	3	2	4	4	0	3
r2	1	0	3	0	4	0	3	3	1
r3	3	4	2	0	1	0	0	4	0
r4	0	1	1	2	0	0	0	1	4
r5	0	2	0	1	3	1	0	2	2
r6	0	0	4	0	0	0	0	0	0
r7	2	0	0	0	0	2	2	0	0
r8	0	0	0	4	0	3	1	0	0

	q
r1	2
r2	0
r3	3
r4	0
r5	1
r6	0
r7	0
r8	0

An example

$$p(q) * p(o)$$

	o1	o2	o3	o4	o5	o6	o7	o8	o9
r1	4	3	0	3	2	4	4	0	3
r2	1	0	3	0	4	0	3	3	1
r3	3	4	2	0	1	0	0	4	0
r4	0	1	1	2	0	0	0	1	4
r5	0	2	0	1	3	1	0	2	2
r6	0	0	4	0	0	0	0	0	0
r7	2	0	0	0	0	2	2	0	0
r8	0	0	0	4	0	3	1	0	0
*	17	20	6	7	10	9	8	14	8

	y
r1	2
r2	0
r3	3
r4	0
r5	1
r6	0
r7	0
r8	0

- The result set is:
- o2, o1, o8, o5, o6, o7, o9, o4, o3

An example

$p(q) * p(o)$

	o1	o2	o3	o4	o5	o6	o7	o	q
r1	4	3	0	3	2	4	4	0	2
r2	1	0	3	0	4	0	3	3	0
r3	3	4	2	0	1	0	0	4	3
r4	0	1	1	2	0	0	0	1	0
r5	0	2	0	1	3	1	0	2	1
r6	0	0	4	0	0	0	0	0	0
r7	2	0	0	0	0	2	2	0	0
r8	0	0	0	4	0	3	1	0	0
*	17	20	6	7	10	9	8	14	

Same order!

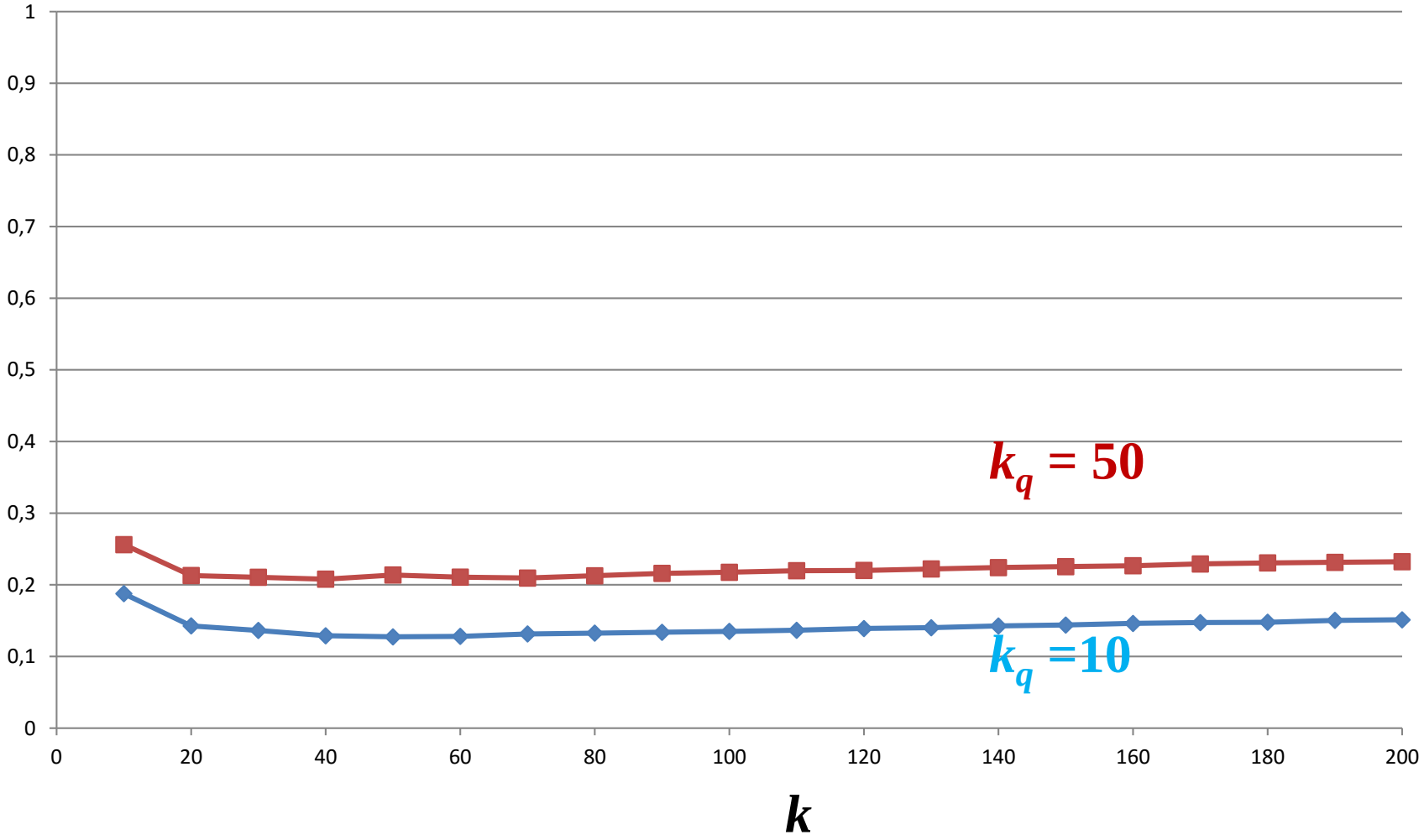
2
1
8
5
6
7
9
4
3

- The result set is:
- o2, o1, o8, o5, o6, o7, o9, o4, o3

Cophir application

- We indexed the CoPhIR dataset, which consists of **106 millions images**, taken from Flickr (www.flickr.com), described by MPEG-7 visual descriptors.
- **Three fields** one for different combination of MPEG-7 descriptors for the same image:
  - Colors: SCD, CSD, and CLD
  - Textures: EHD and HTD
  - ALL: SCD, CSD, CLD, EHD, and HTD
- **Textual field** metadata: title and tags
- 20,000 reference objects, $k_i = 50$, $k_q = 10\text{-}50$
- Lucene inverted space occupation about 50GB

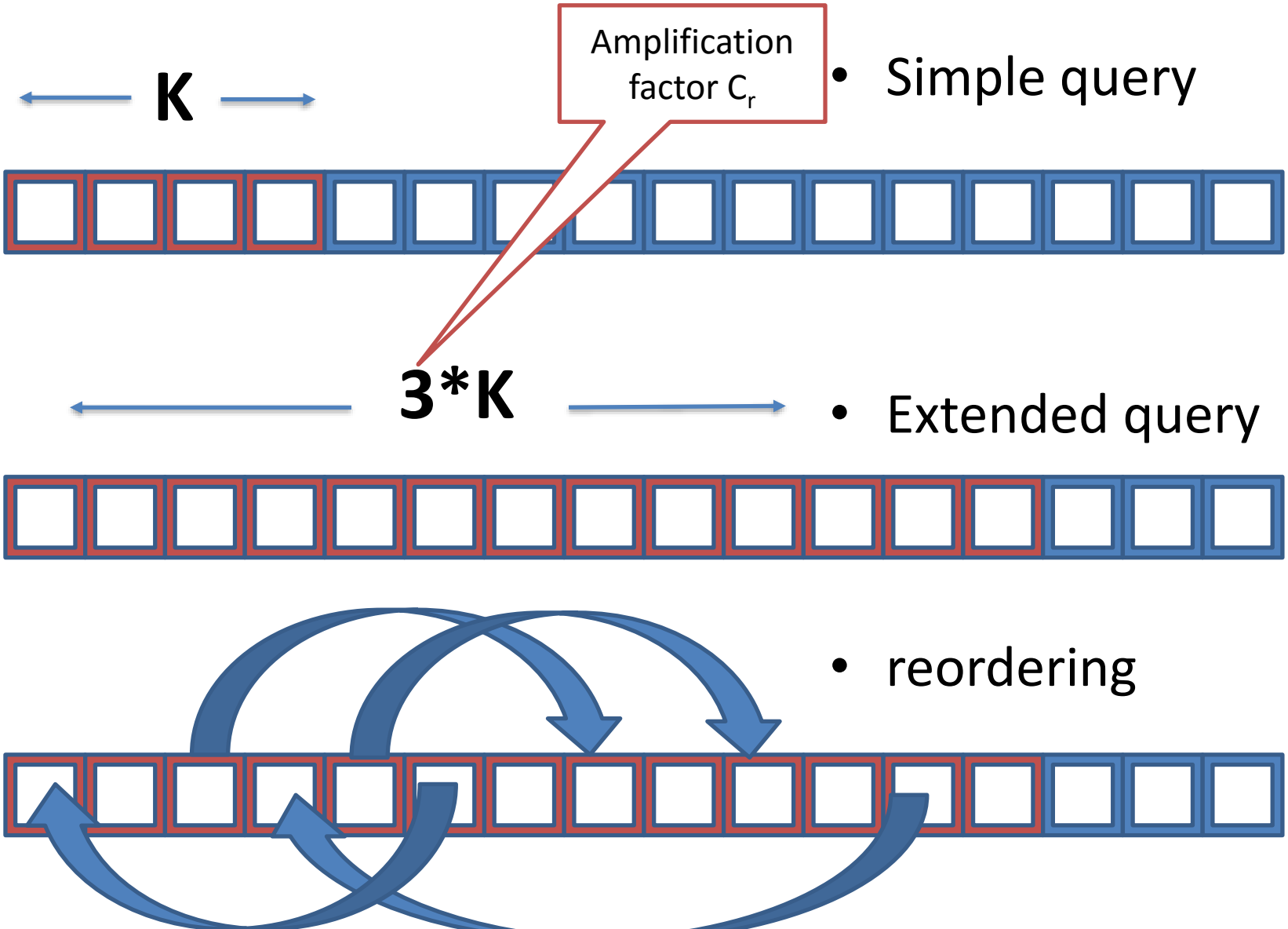
Recall *without* reordering



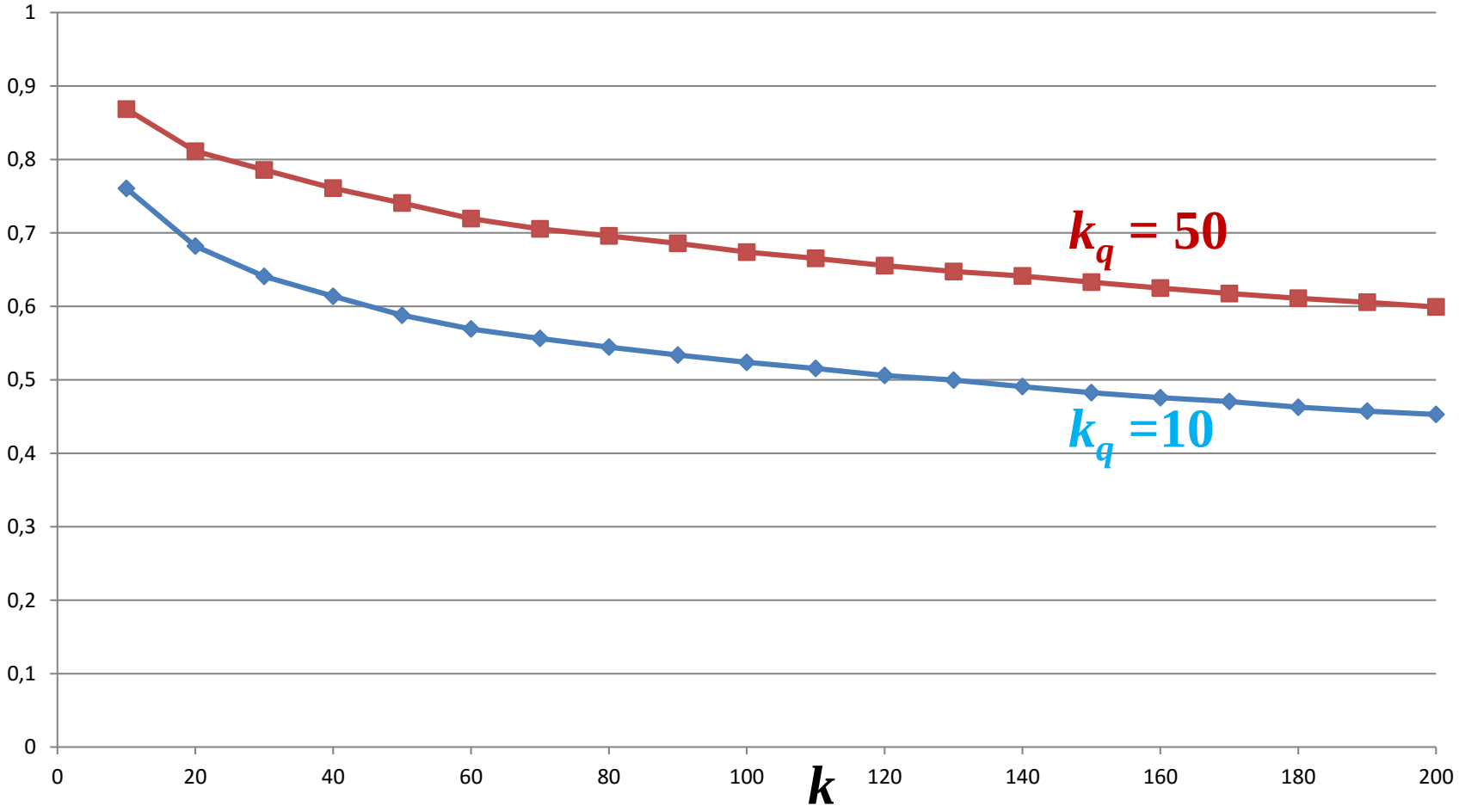
Query extension and reordering

- Instead searching for just top k images, we extend the query to more results (i.e., top 2000 images)
- and we use the actual similarity function to reorder the result set.
- The impact of this extension on the efficiency is negligible, provided that we have can fast randomly access the DB of the features associated with the images.

Query extension and reordering



Recall *with* reordering



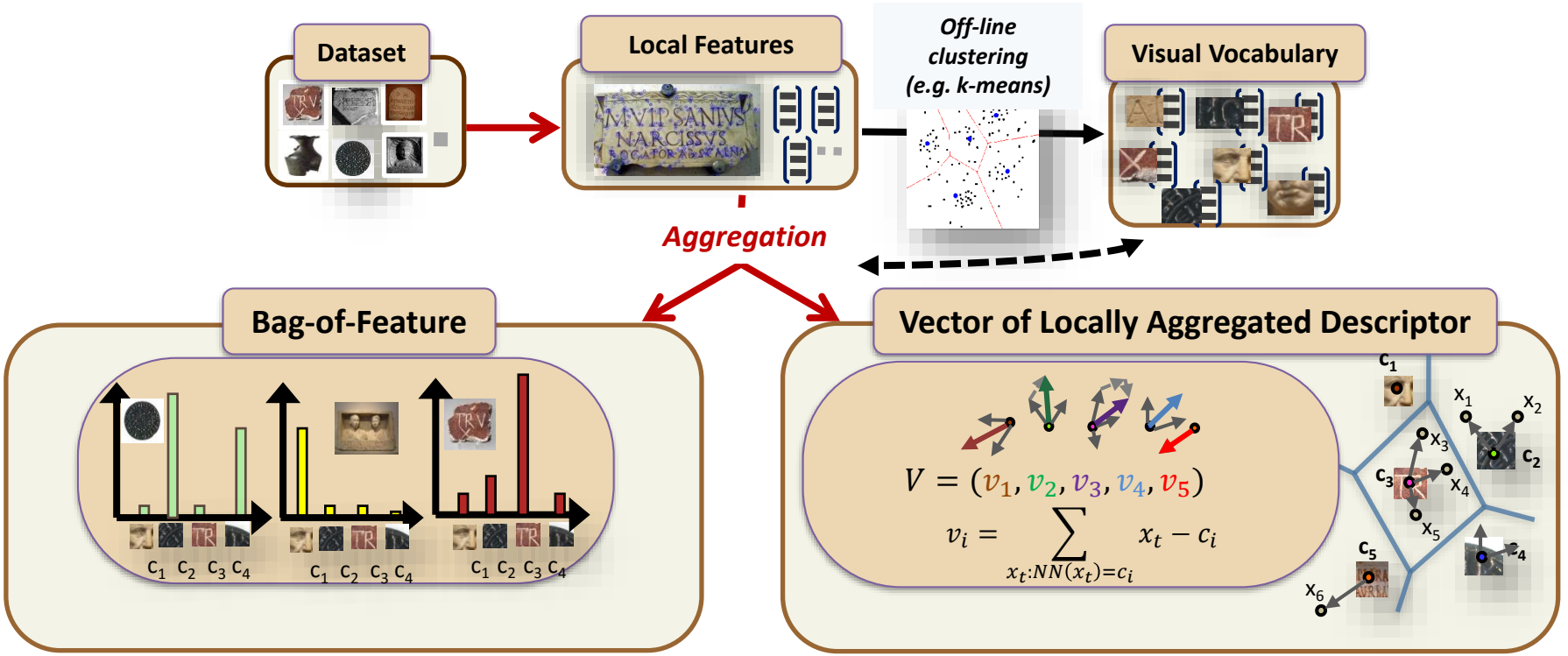
EXPLOITING VECTOR FEATURES

VECTOR OF LOCALLY AGGREGATED DESCRIPTOR (VLAD)

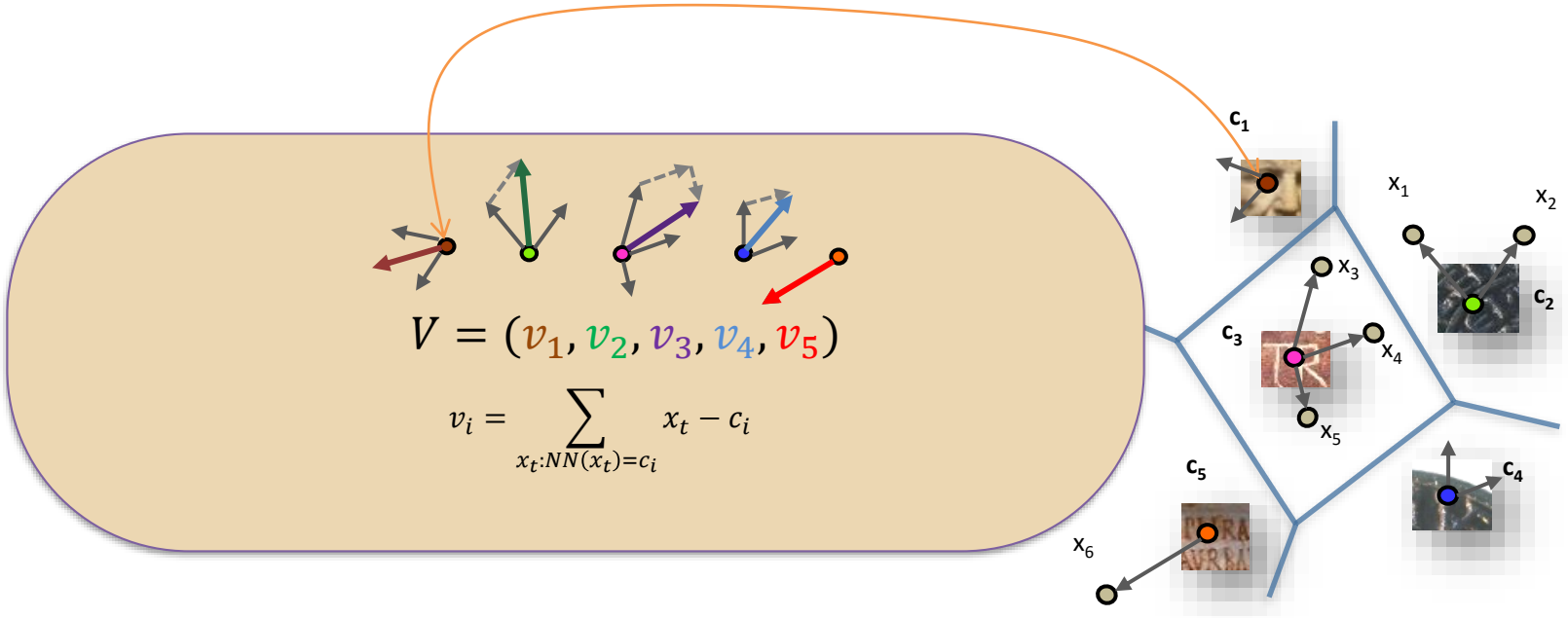
Blockwise Permutation-based Approach

- We want to get rid of the reordering phase!
- We exploit the internal representation of the **Vector of Locally Aggregated Descriptor (VLAD)**

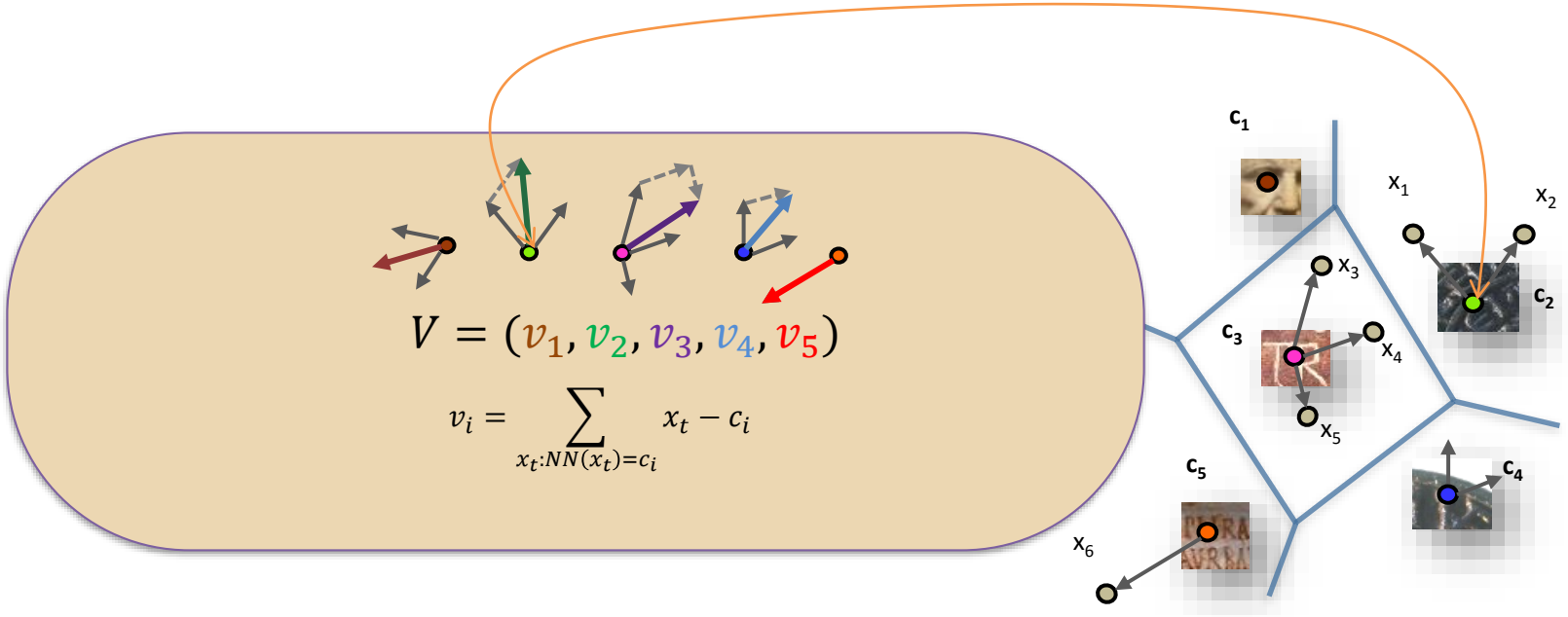
Aggregation Methods



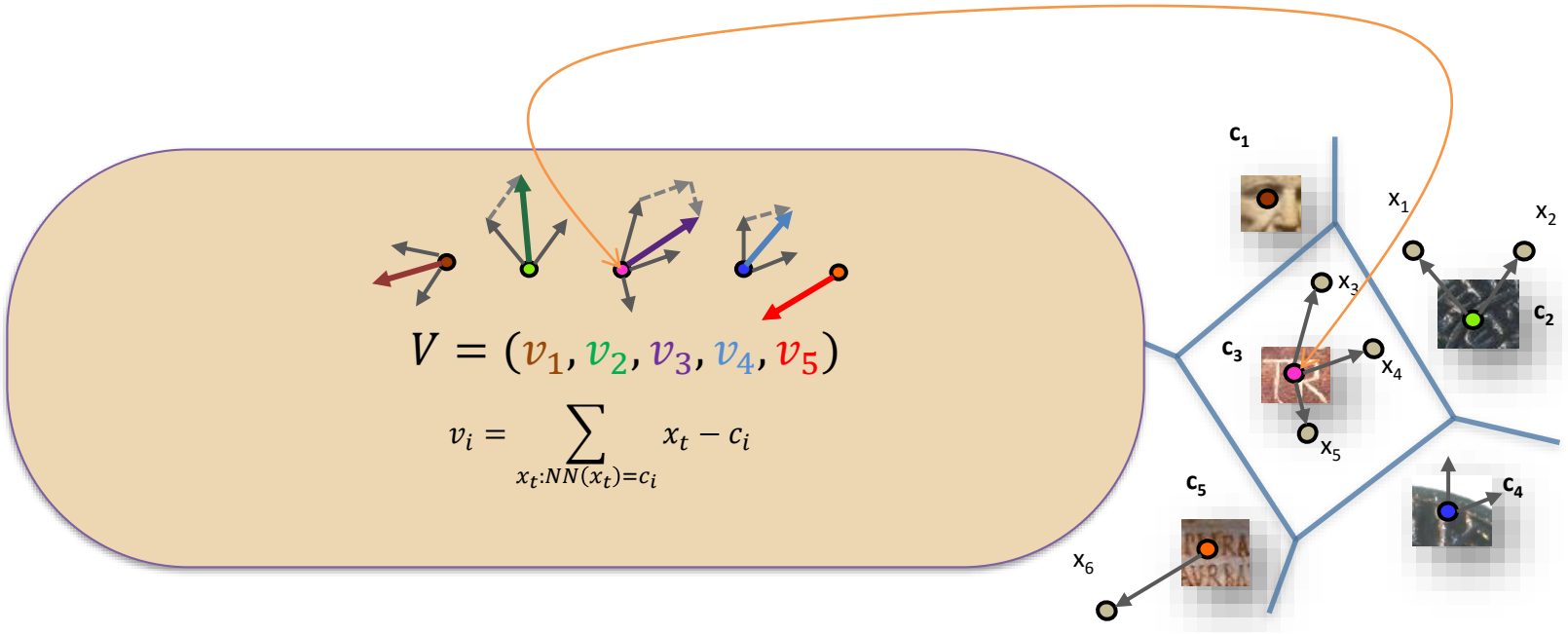
Vector of Locally Aggregated Descriptor



Vector of Locally Aggregated Descriptor

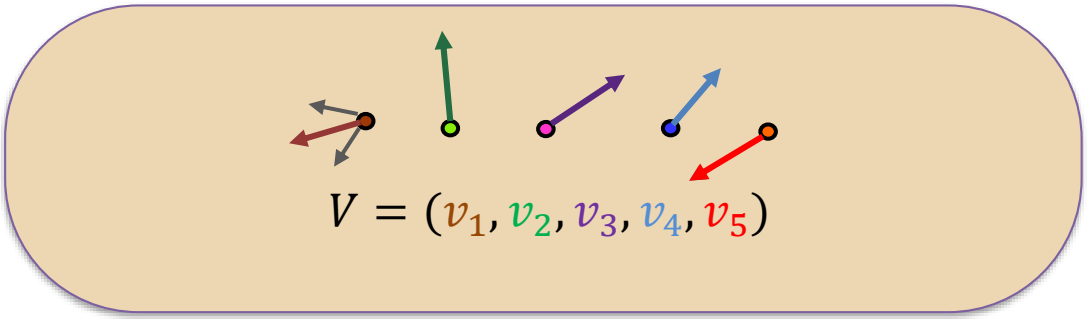


Vector of Locally Aggregated Descriptor

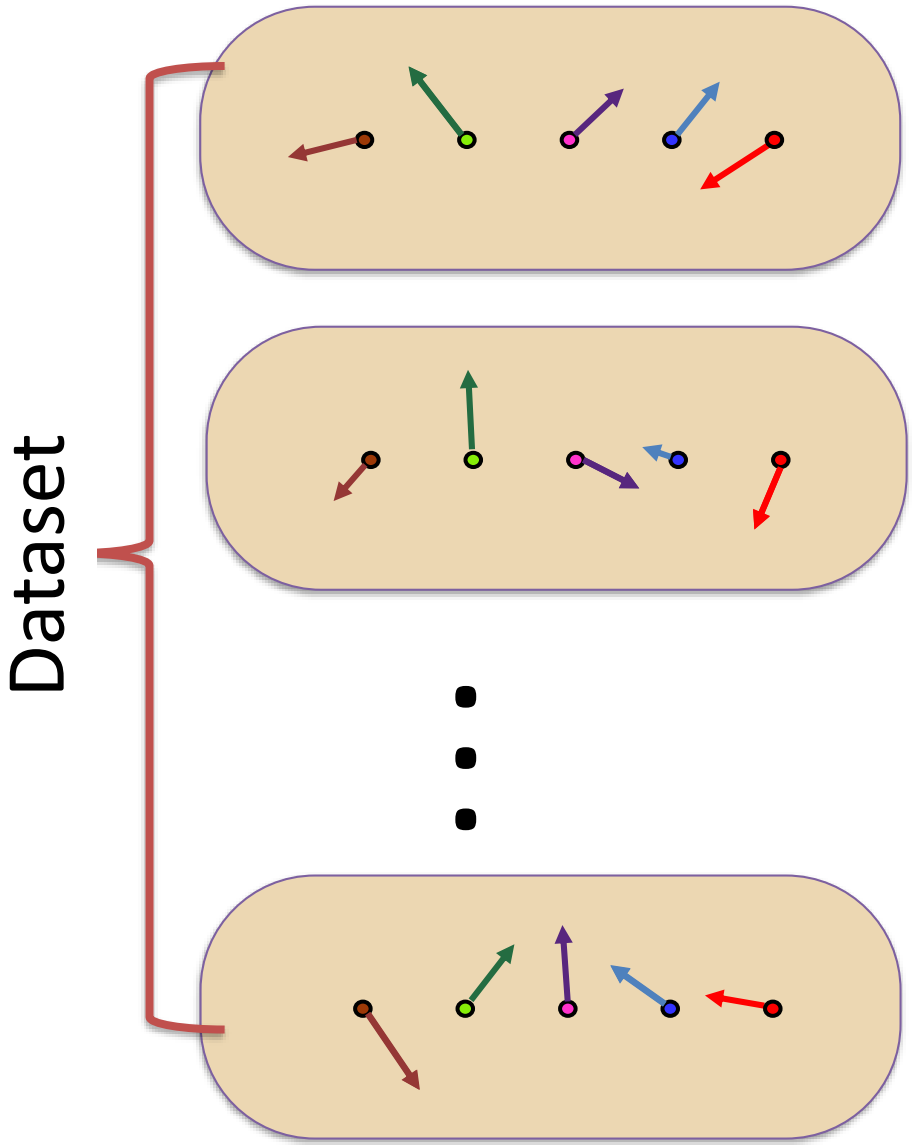


Blockwise Permutation

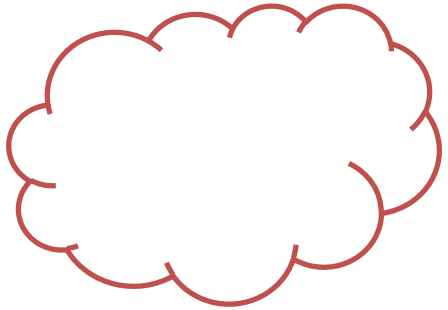
- We represent each «subvector» as a with a SRT (ie. a permutation of random reference subvectors)



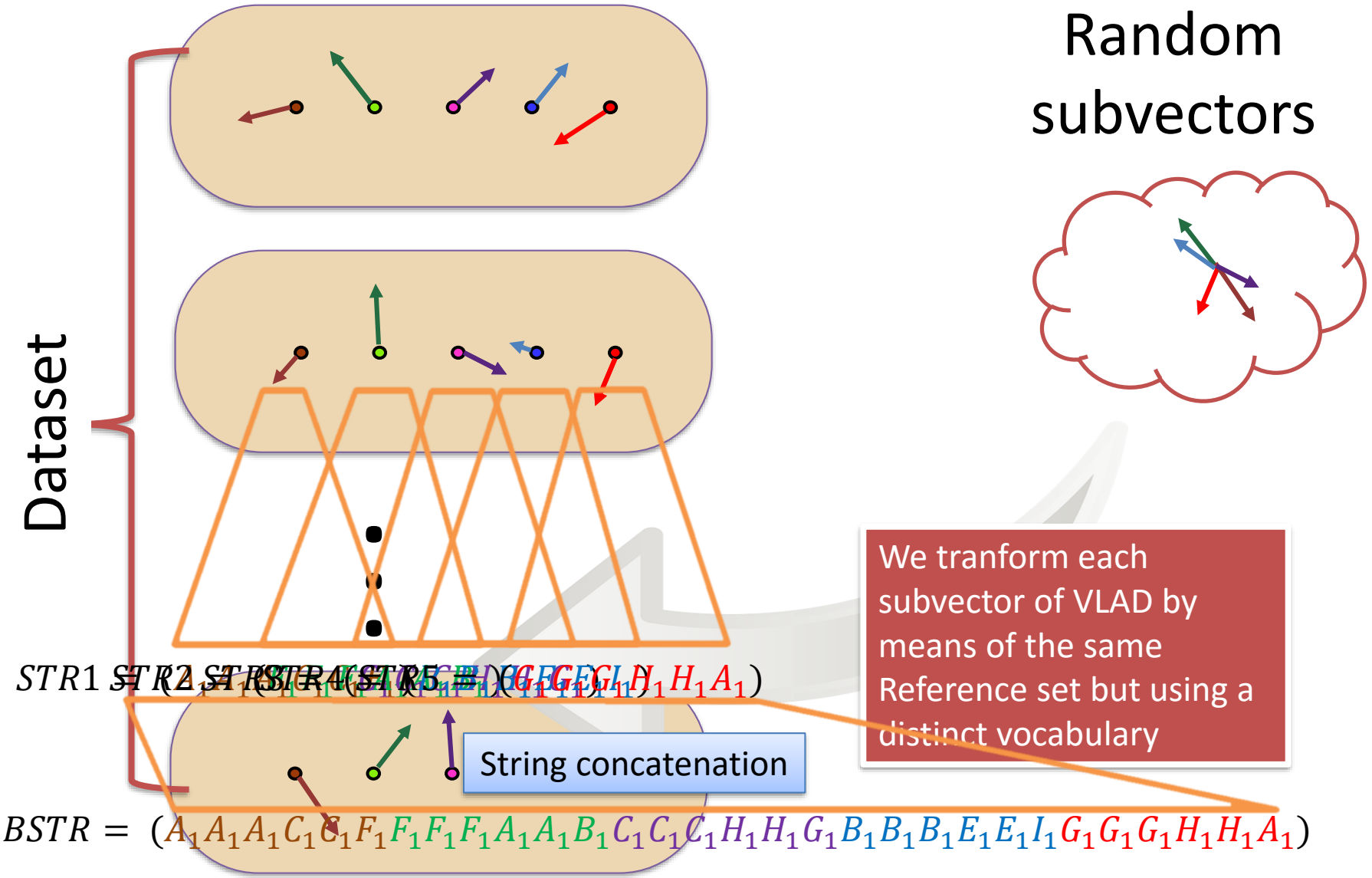
Blockwise Permutation



Random subvectors

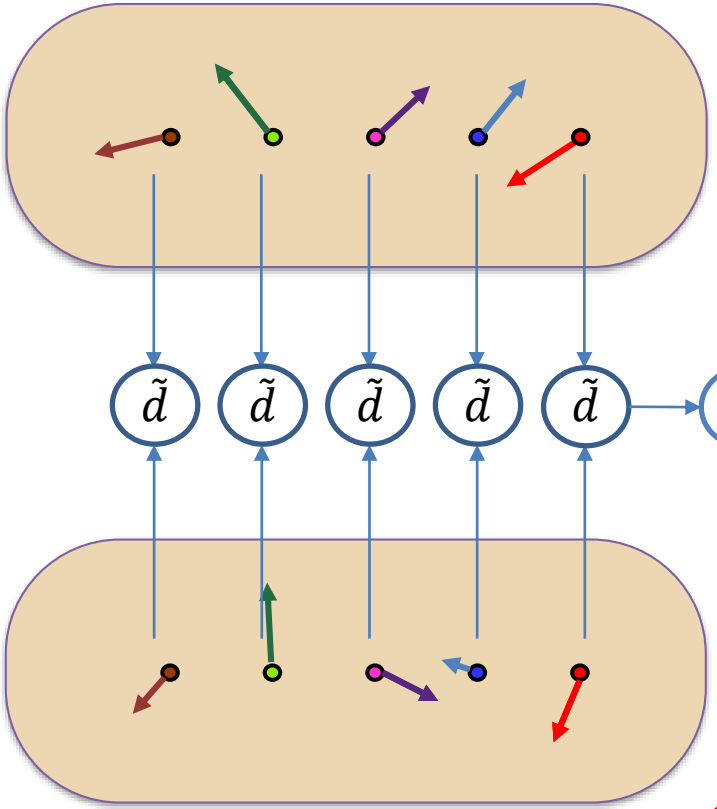


Blockwise Permutation



Blockwise Surrogate Text Representation (BSRT)

$V = (v_1, v_2, v_3, v_4, v_5)$



$$\tilde{d}(V,W)^2 = \sum_1^K \tilde{d}(v_i, w_i)^2 = \sum_1^K \|p(v_i) - p(w_i)\|^2$$

$$p(v_i) * p(w_i) = \alpha - \beta \tilde{d}(V,W)^2$$

$W = (v_1, v_2, v_3, v_4, v_5)$

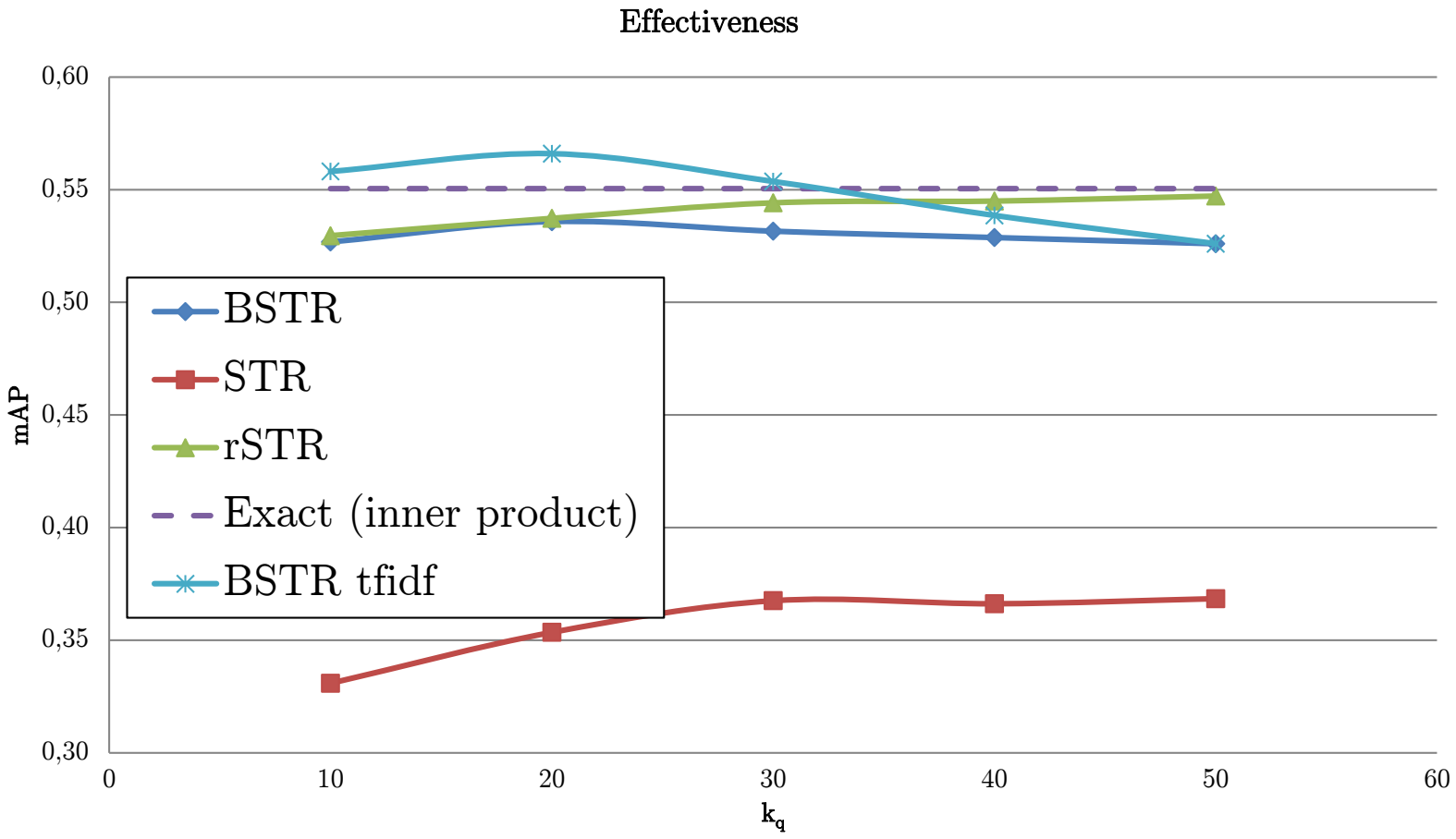
Experiments

- Datasets
  - INRIA Holidays (J´egou et al., 2010a; J´egou et al., 2012) is a collection of 1,491 holiday images. The authors selected 500 queries and for each of them a list of positive results.
  - Distraction set Holidays 1 million dataset the Flickr1M.

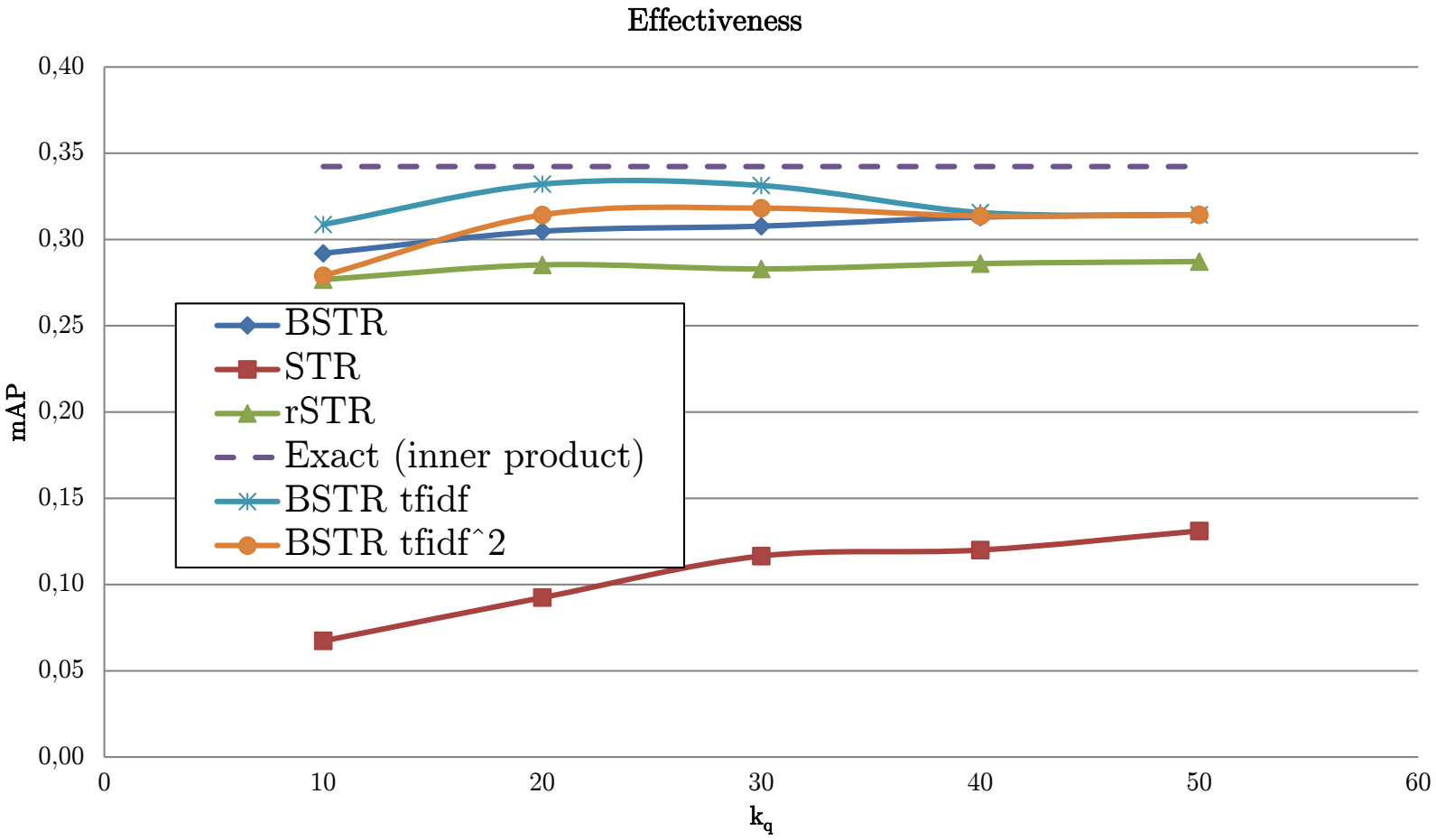
Indexing

- We wanted to compare against the baseline STR approach,
- VLAD exact (Inner product)
- Lucene index:
  - Baseline (STR): 4,000 reference objects (VLAD vectors considered as a whole, scalar prod), k_x = 50
  - Baseline + Reordering (rSTR): reordering of the first 1K result set of LUCENE using scalar prod, k_x = 50
  - BSTR: 20,000 reference objects (VLAD subvectors), k_x = 50
  - BSTR tfidf: query reduction keeping the first best tfidf terms
  - BSTR tfidf^2: query reduction + index reduction keeping the first best tfidf terms

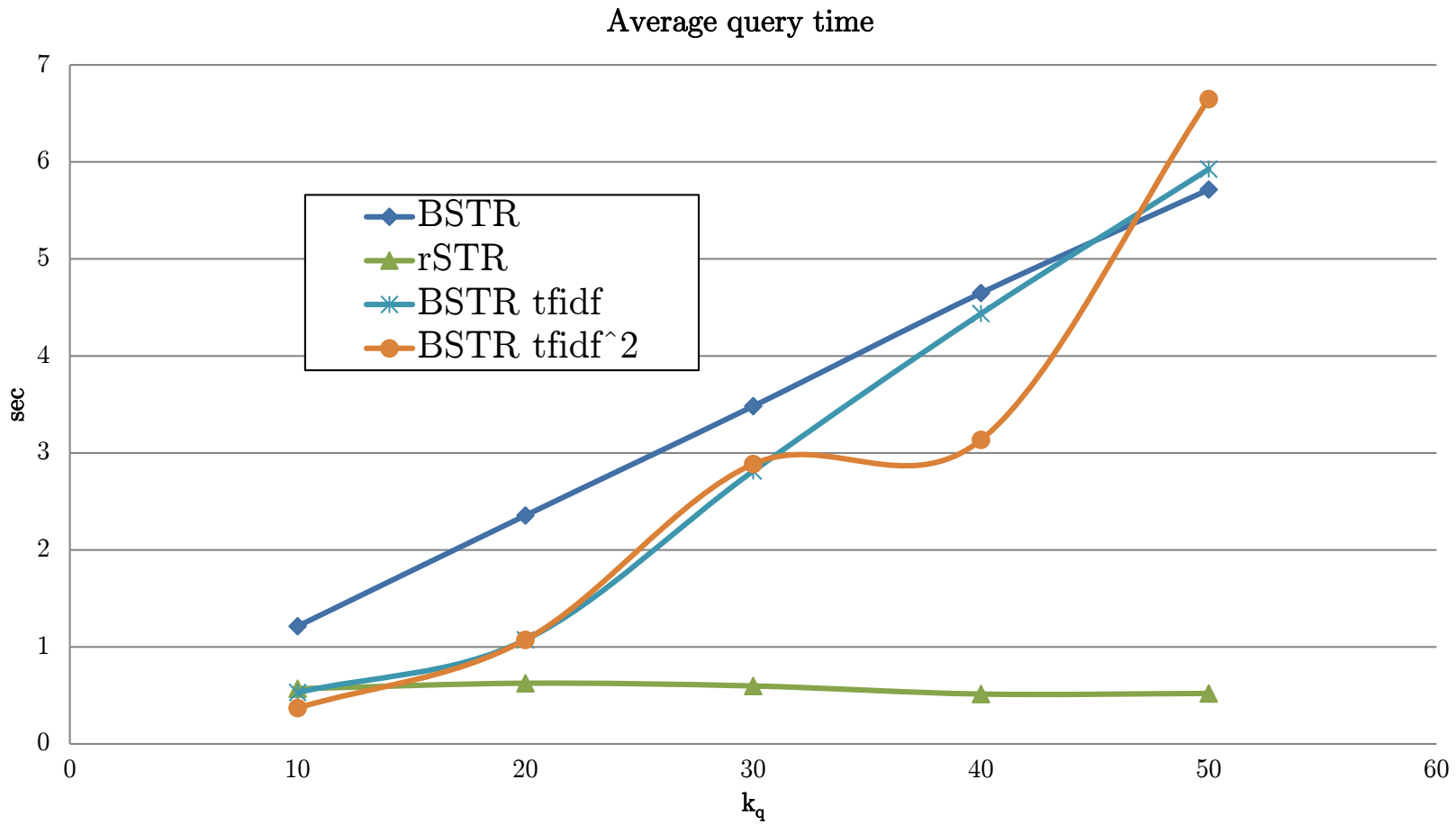
Effectiveness (1,491 images)



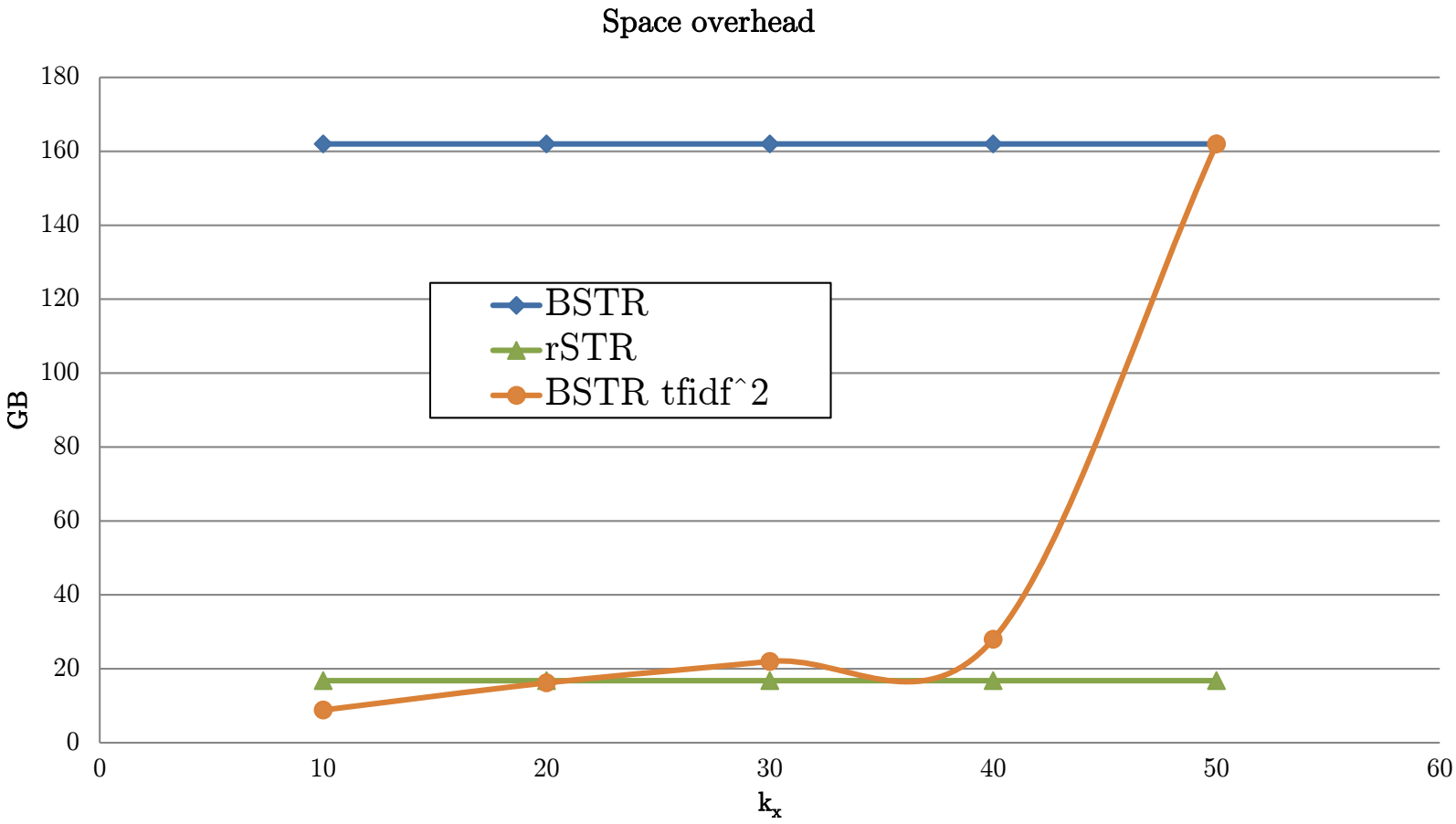
Effectiveness (1,491+1M images)



Query time


Average query time
sec
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60
k_q
BSTR
rSTR
BSTR tfidf
BSTR tfidf^2

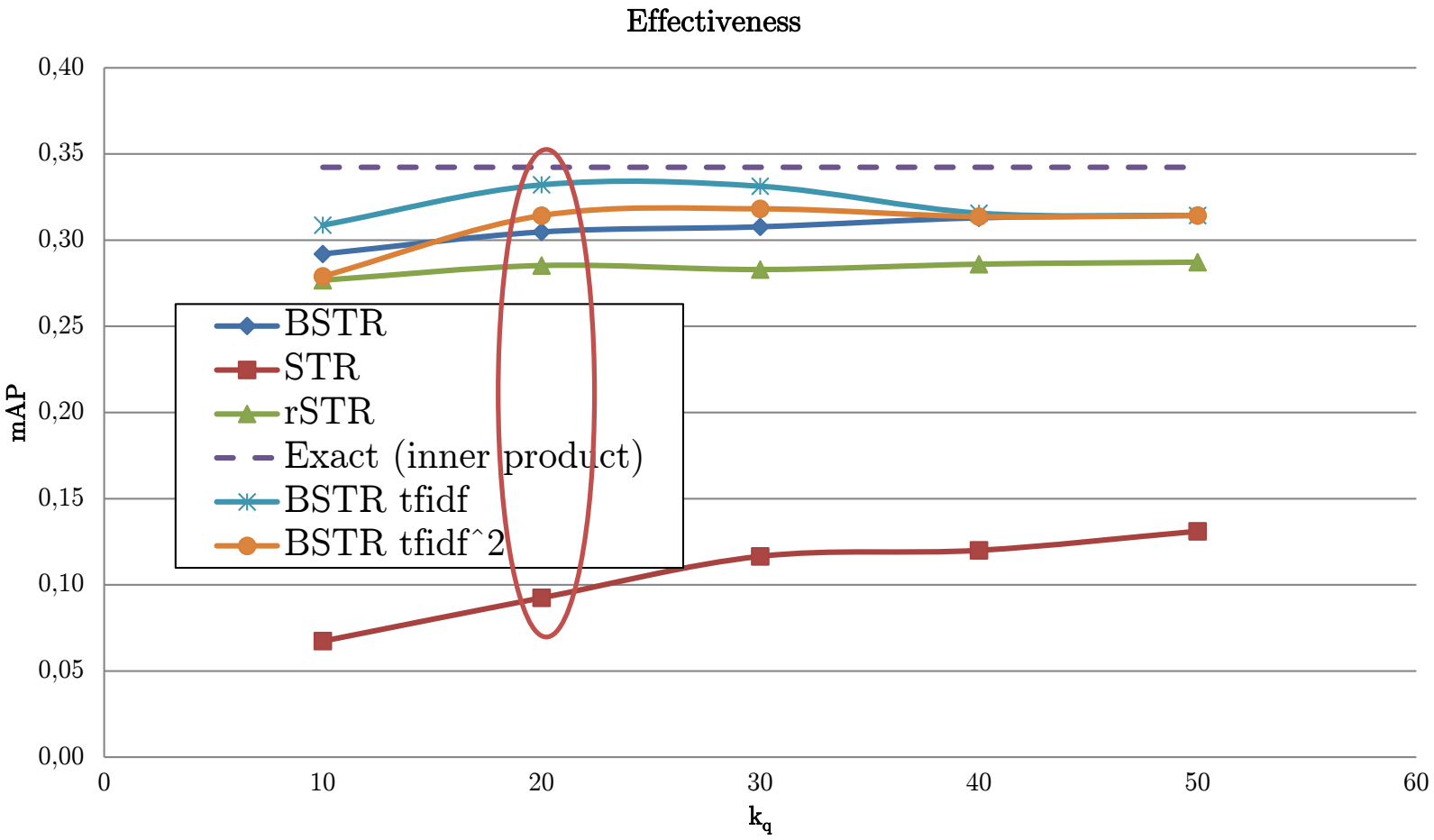
Space overhead



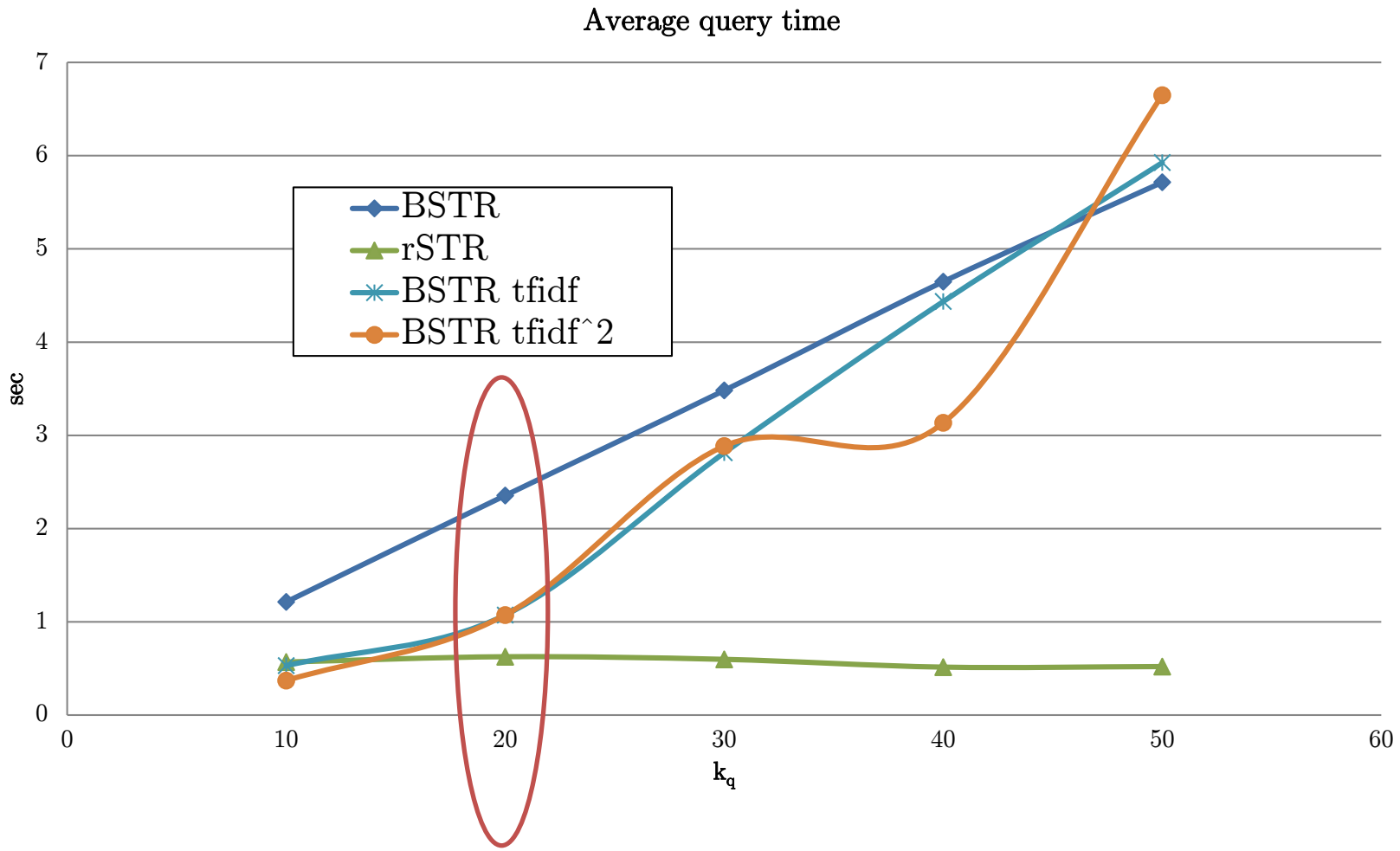
And the winners is?

- There is no 'winner' for all situations
- However, tfidf^2 for $k_x=k_q=20$ represents a good compromise in terms of effectiveness, efficiency and space overhead.

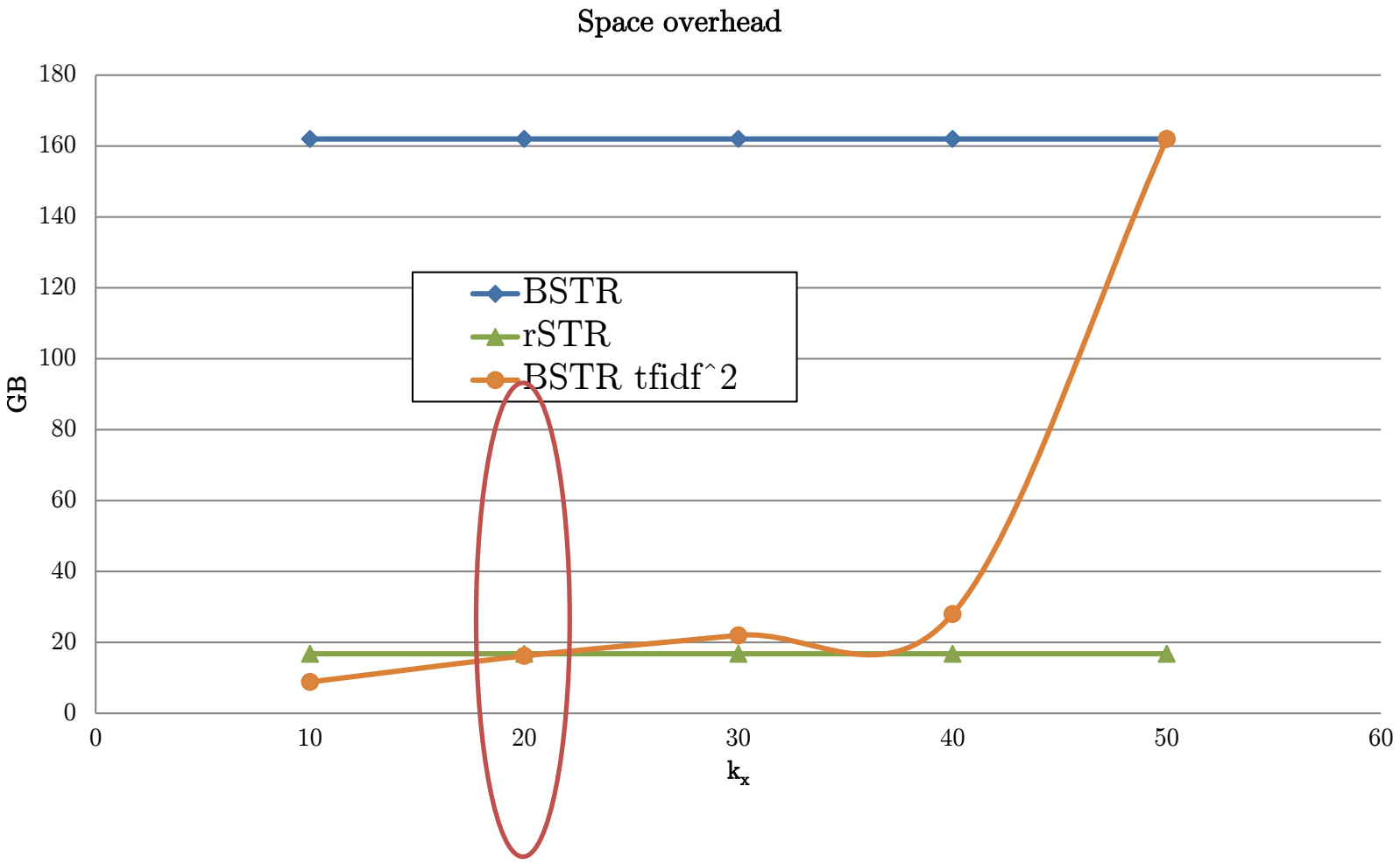
Effectiveness (1,491+1M images)



Query time


Average query time
sec
7
6
5
4
3
2
1
0
0
10
20
30
40
50
60
k_q
BSTR
rSTR
BSTR tfidf
BSTR tfidf^2

Space overhead



Conclusion

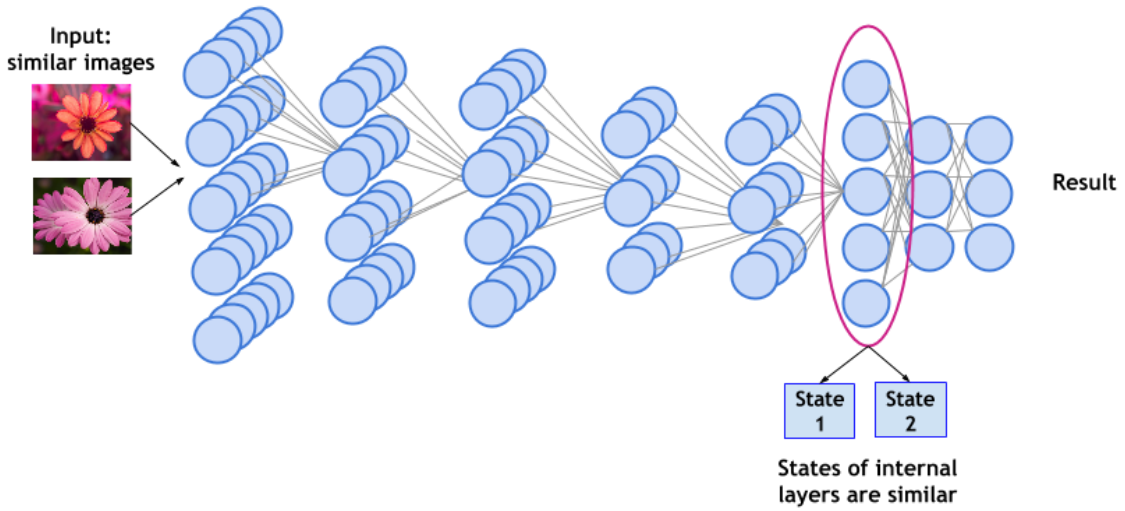
- Blockwise is a generalization of STR / permutation specific for vectors
- Advantages:
  - Get rid of reordering
- Disadvantages:
  - Expansion of the dictionary, very big index comparing with the standard STR
  - We tried to mitigate with the use of tf*idf but this implies a double indexing phase.

DEEP CONVOLUTIONAL NEURAL NETWORK FEATURES

Deep Convolutional Neural Network features

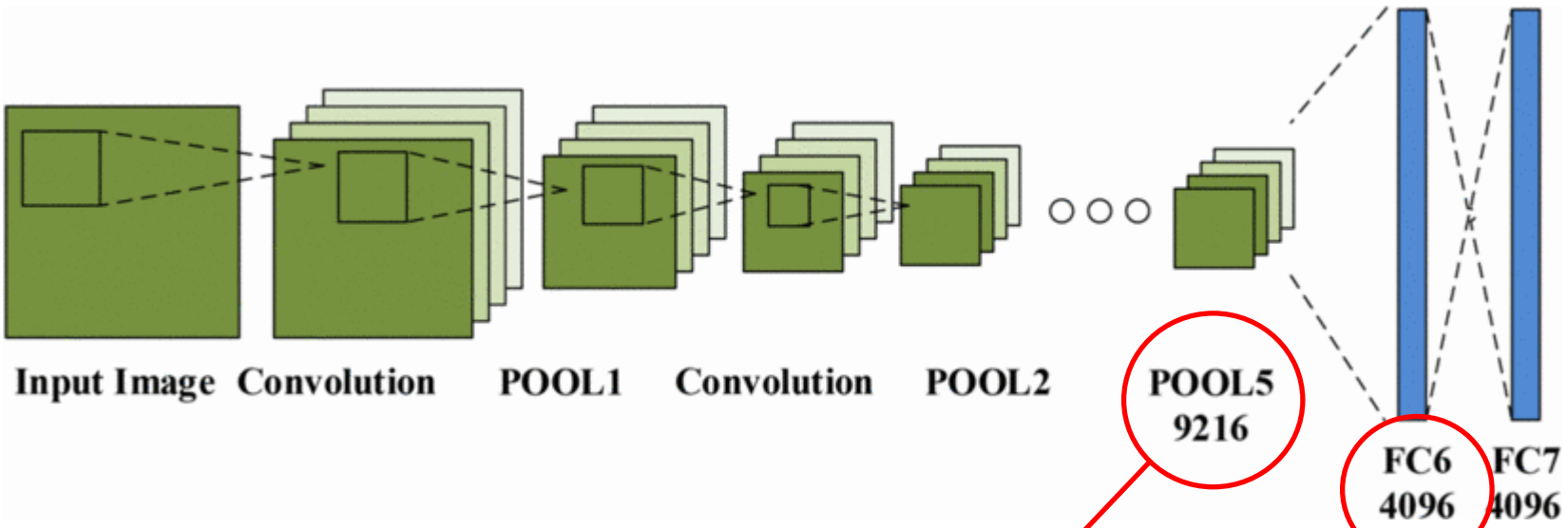
- Deep CNN are CNN composed of several layers.
- A well-known CNN is AlexNet.
  - AlexNet was trained to recognize 1000 concepts from ImageNet

- To extract visual features from training images, we can use the **Caffe** framework



- Deep CNN features
  - If two similar images are given in input to the AlexNet, the states of internal layers of the net are similar as well.

AlexNet



using the rectified linear units (ReLUs) activation functions 90% of the component of these vectors are 0.0

Credit: Zhu, Haigang, et al. "Orientation robust object detection in aerial images using deep convolutional neural network." *Image Processing (ICIP), 2015 IEEE International Conference on*. IEEE, 2015.

Exploiting sparsity of DCNN Features

- Canonical approach for searching DCNN Features is to use the Euclidean distance (or dot product) on L2-normalized vectors to the unit length
- However, since most of DCNN features vectors are 0s,
- we propose to index them with a inverted file of a text-search engine

Exploiting sparsity of DCNN Features

- If vectors are L2-normalized to the unit length:

$$\|\boldsymbol{x} - \boldsymbol{y}\|^2 = 2(1 - \boldsymbol{x} * \boldsymbol{y})$$

- We have again the monotonic relationship between L2 and x*y
- First, we quantize the vector components

$$\boldsymbol{x} = (x_1, \ldots, x_n) \rightarrow x_Q = (\lfloor Q x_1 \rfloor, \ldots, \lfloor Q x_n \rfloor)$$

Exploiting sparsity of DCNN Features

- For instance:

$$\text{Q=30, } \boldsymbol{x} = (0.01, 0.15, 0.09) \rightarrow (0, 4, 2)$$

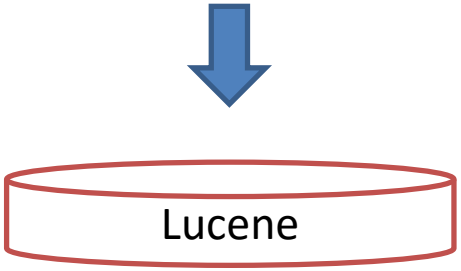
- For component $x_i < \frac{1}{Q}$ $\quad\quad \lfloor Q x_i \rfloor \rightarrow 0$
- which increases the sparsity of about 3-4%

Exploiting sparsity of DCNN Features

- As usual, we create a code-book of vector components by associating to each component x_i a unique term f_i:
- $doc(\boldsymbol{x}) = \text{" ... } f_i\ f_i\ f_i\ f_i \text{ ... } f_j\ f_j\ f_j\ f_j \text{ ..."}$

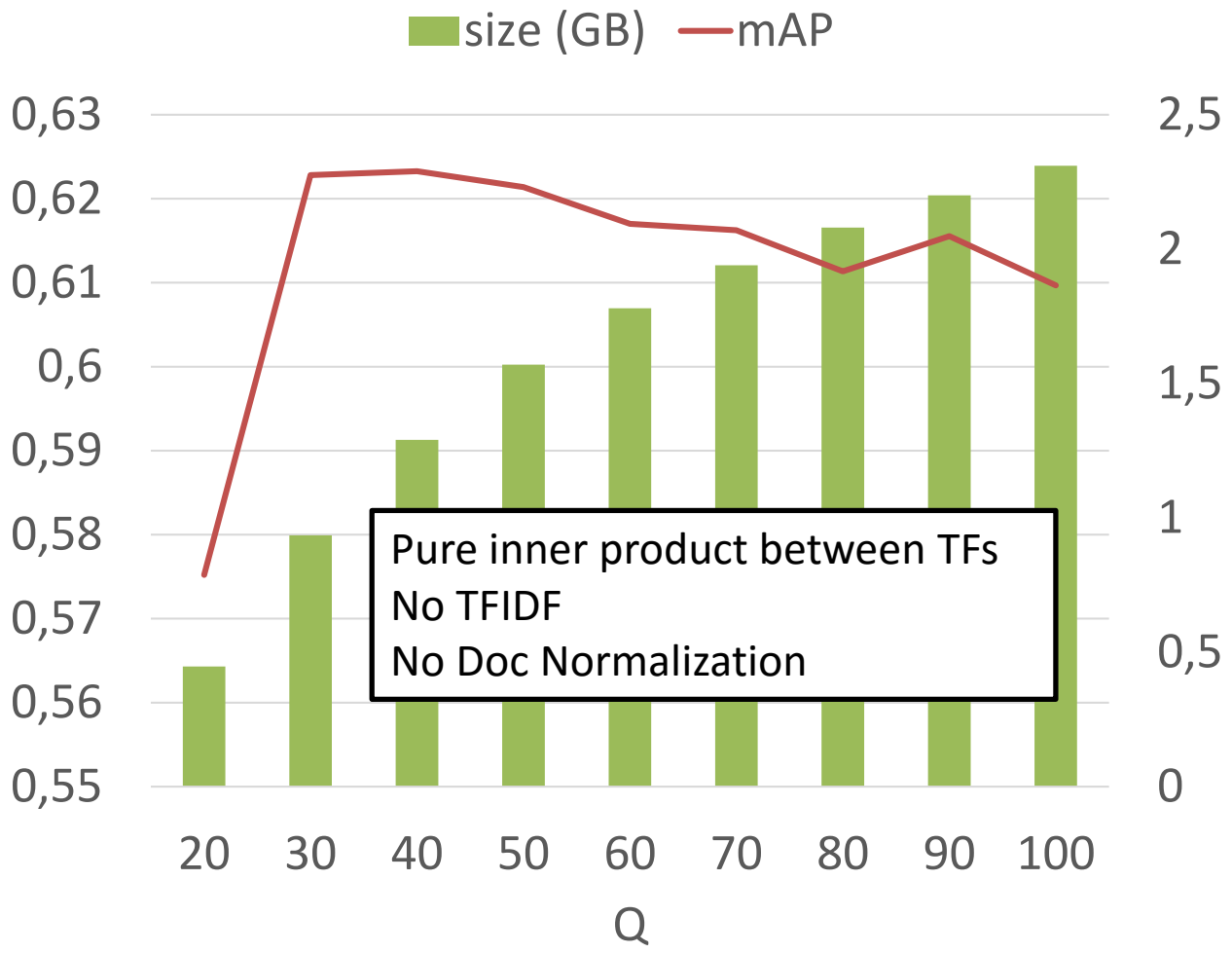
$\lfloor Qx_i \rfloor$ times

$\lfloor Qx_j \rfloor$ times



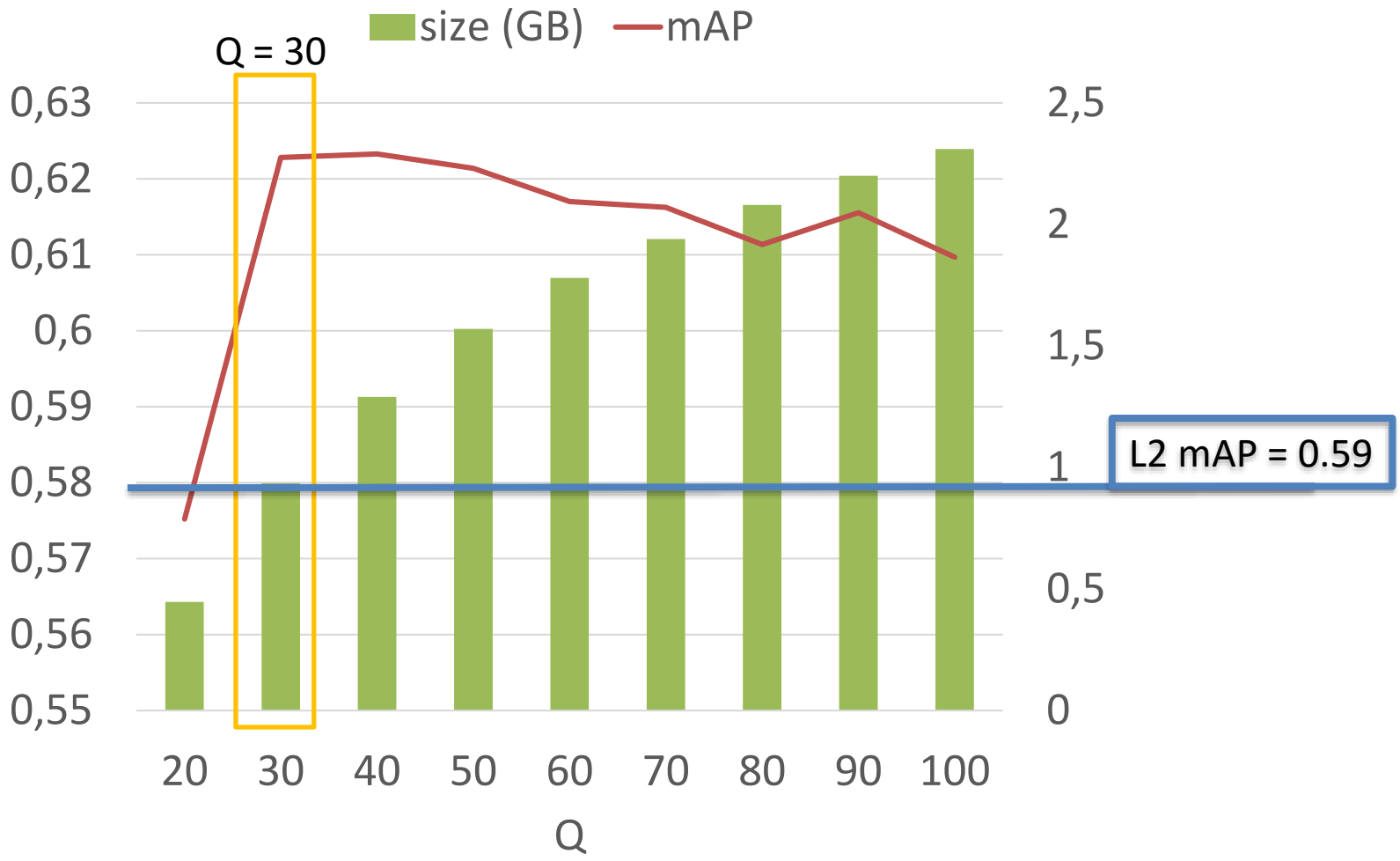
mAP and quantization factor Q

- Holidays 1 + million dataset the Flickr1M & CNN FC6



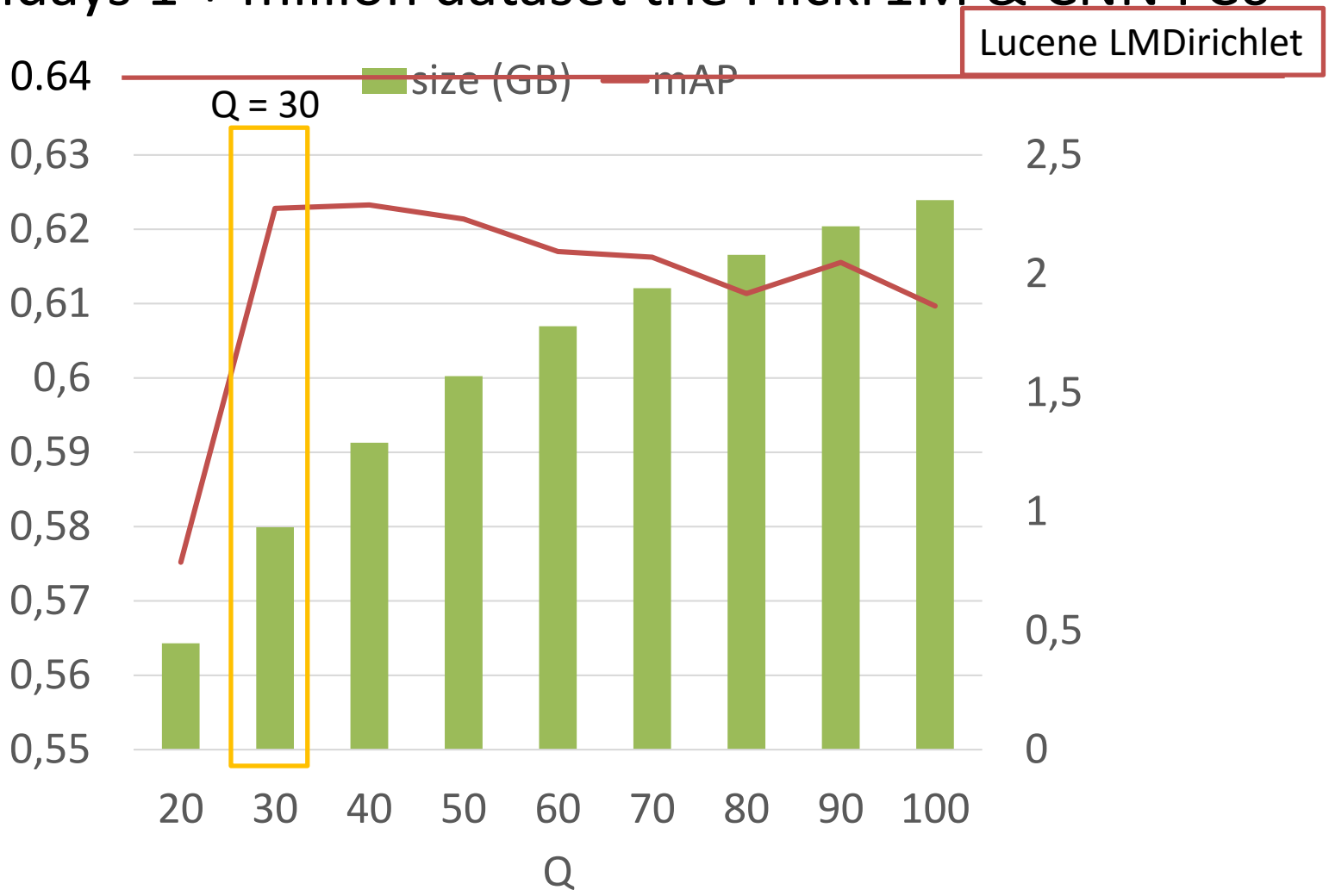
mAP and quantization factor Q

- Holidays 1 + million dataset the Flickr1M & CNN FC6

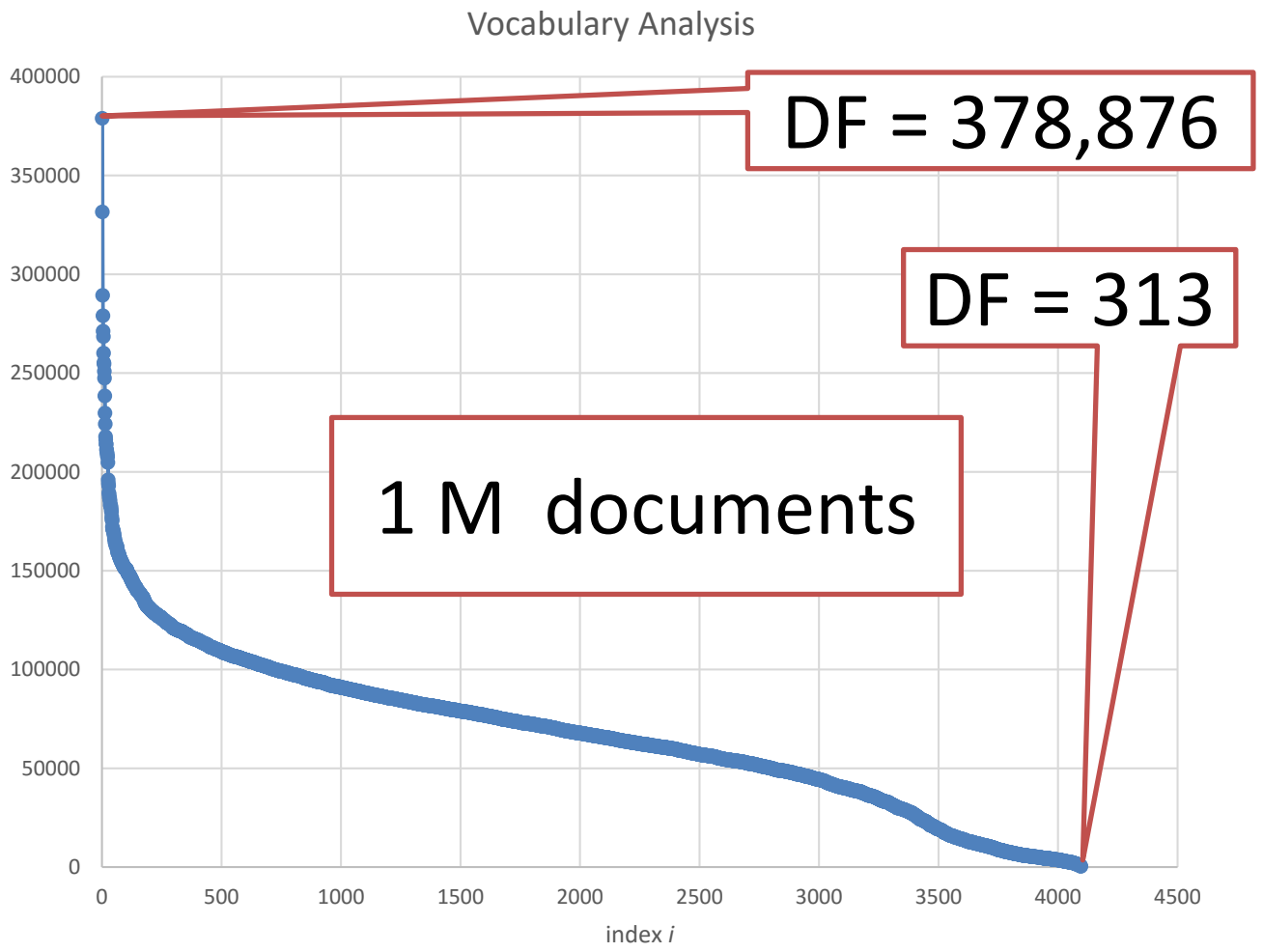


mAP and quantization factor Q

- Holidays 1 + million dataset the Flickr1M & CNN FC6



Vocabulary Analysis



Query Reduction

- The observation about terms document frequency leads to the idea of using tfidf to reduce the query length by cutting off terms with lower tfidf weight.
- Since in inverted files the query time is usually proportional with the length of the query, this approach give a great improvement in terms of response time.
- Queries in Lucene are in the form:
  - f_{343}^1 f_{241}^2 f_{938}^3 f_{455}^2 … f_{533}^3

Query Reduction

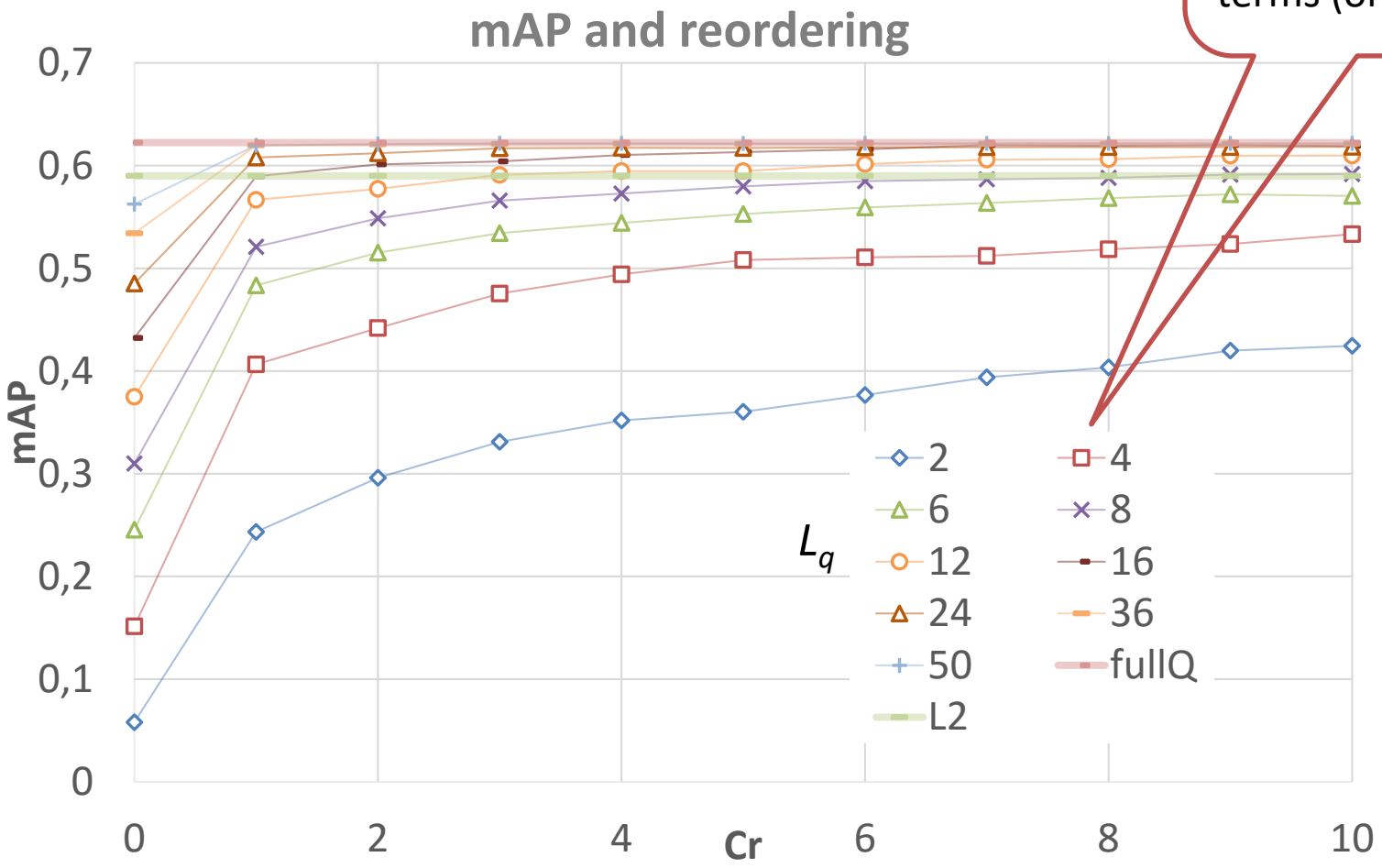
- The observation about terms document frequency leads to the idea of using tfidf to reduce the query length by cutting off terms with lower tfidf weight.
- Since in inverted files the query time is usually proportional with the length of the query, this approach give a great improvement in terms of response time.
- Queries in Lucene are in the form:
  - f_{123}^2 f_{221}^3 f_{28}^2 f_{1233}^2 … f_{753}^3

tfidf	8.23	7.5	7.1	6.9

mAP with Query Reduction and Reordering

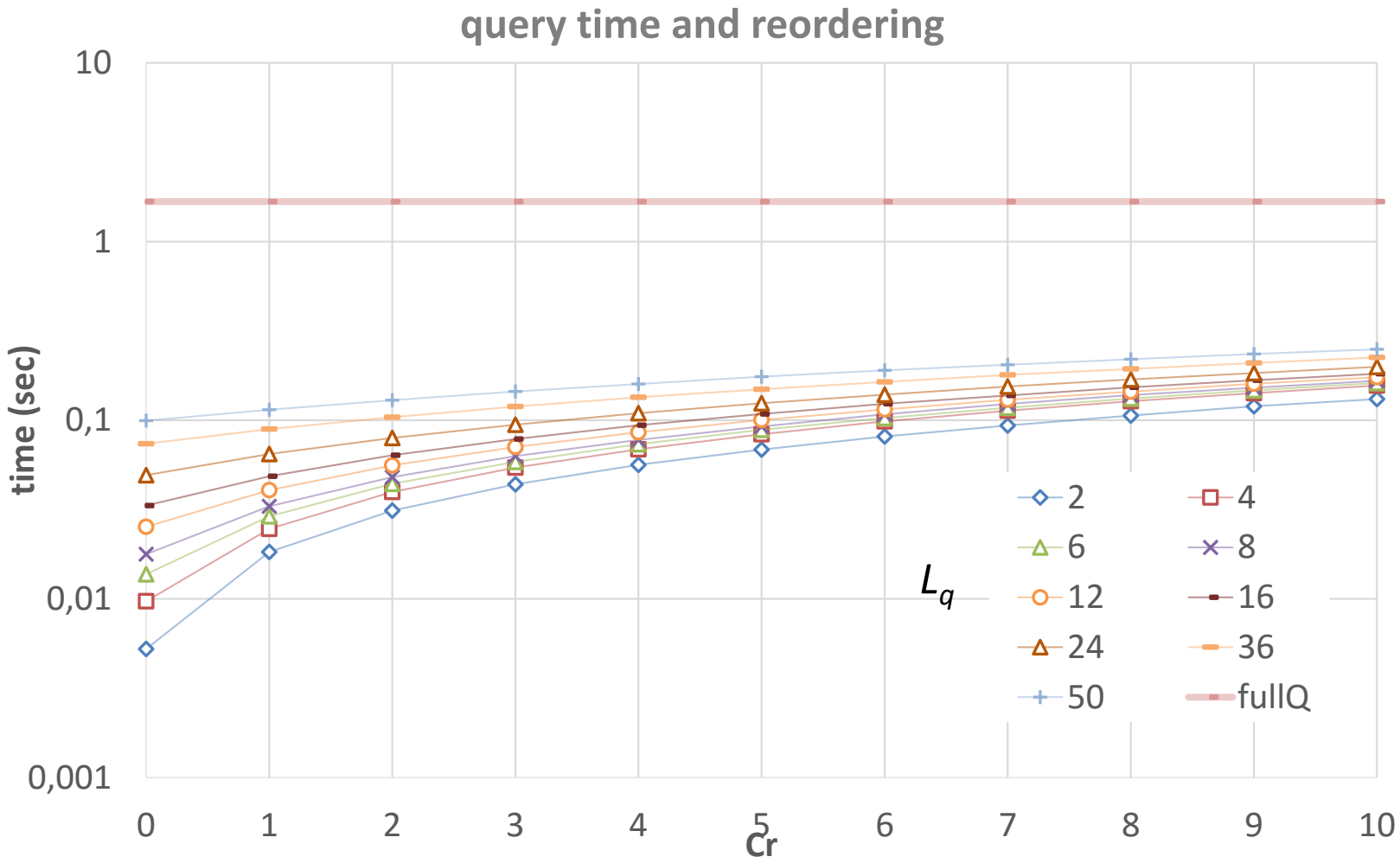
$C_r = 0$ no reordering means

L_q is the query length w.r.t a full query of about 275 terms (on average)



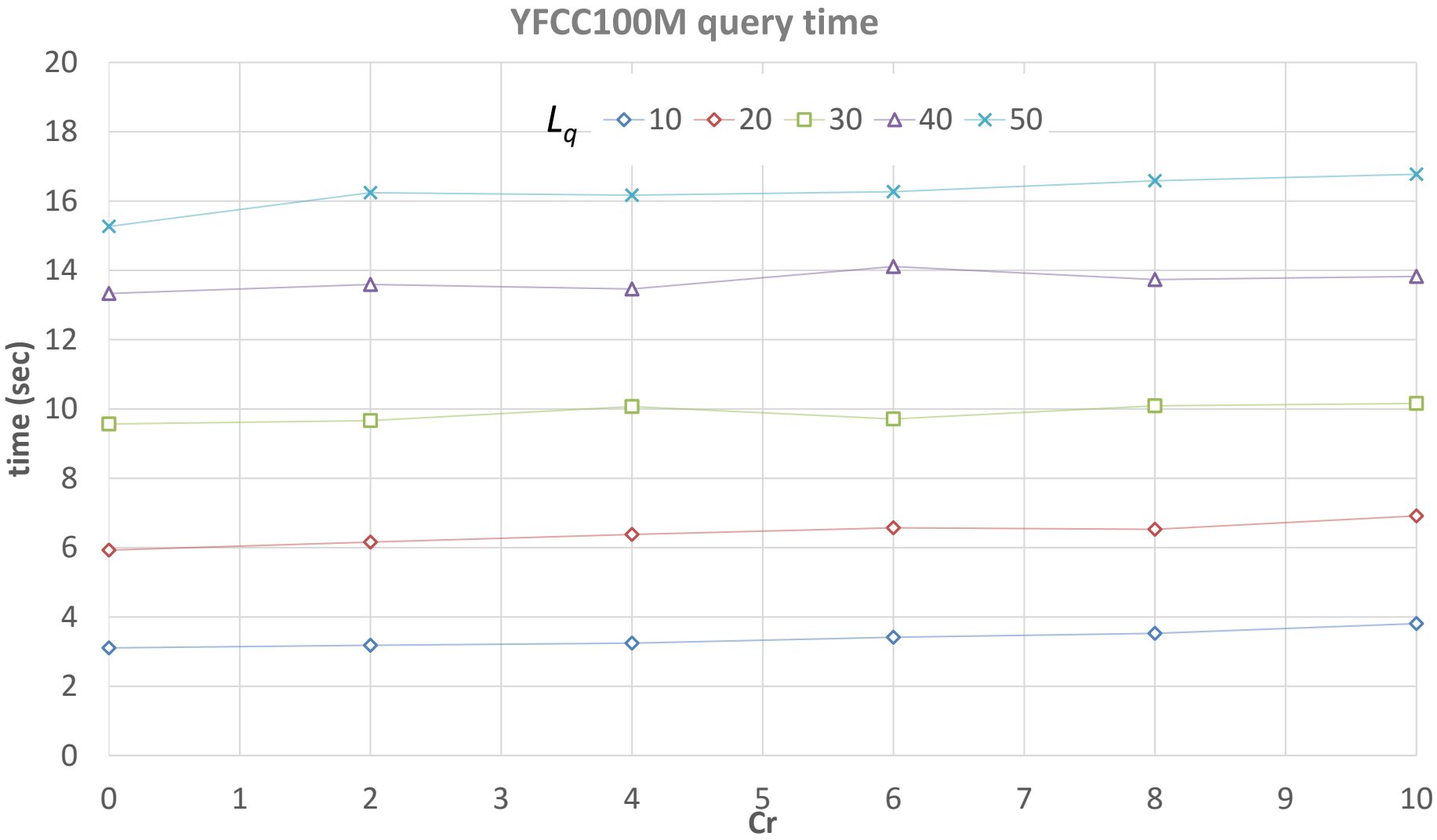
Query time with Query Reduction and Reordering

C_r = 0 no reordering means



The YFCC100M dataset

Yahoo Flickr Creative Commons 100 Million images



Pro and cons

- Pro:
  - straightforward text transformation, which does not demand costly elaborations, as the permutation-based approaches.
  - It allow us to tune the query costs versus the quality of the approximation by specifying the length of the query
  - no need of maintaining the original features for reordering the result set.
- Cons:
  - Specifically built for sparse DCNN features

Conclusion

- We proposed a various efficient approach to build a content-based image retrieval on top of a text search.
- Advantages:
  - Little implementation effort
  - Strong support
  - Easy Parallelization
  - Combination of textual queries and image similarity comes for free
- Disadvantages
  - Approximate results

References

- Gennaro, Claudio, et al. "An approach to content-based image retrieval based on the Lucene search engine library." Research and Advanced Technology for Digital Libraries. Springer Berlin Heidelberg, 2010. 55-66.
- Amato, Giuseppe, et al. "Combining local and global visual feature similarity using a text search engine." *Content-Based Multimedia Indexing (CBMI), 2011 9th International Workshop on*. IEEE, 2011.
- Amato, Giuseppe, et al. "Large scale image retrieval using vector of locally aggregated descriptors." *Similarity Search and Applications*. Springer Berlin Heidelberg, 2013. 245-256.
- Amato, Giuseppe, Claudio Gennaro, and Pasquale Savino. "MI-File: using inverted files for scalable approximate similarity search." *Multimedia tools and applications* 71.3 (2014): 1333-1362.
- Amato, G., Bolettieri, P., Falchi, F., Gennaro, C. and Vadicamo, L. Using Apache Lucene to Search Vector of Locally Aggregated Descriptors. In Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2016) - Volume 4: VISAPP, pages 383-392
- Gennaro, Claudio. “Large Scale Deep Convolutional Neural Network Features Search with Lucene”. arXiv:submit/1513728 [cs.CV] 31 Mar 2016