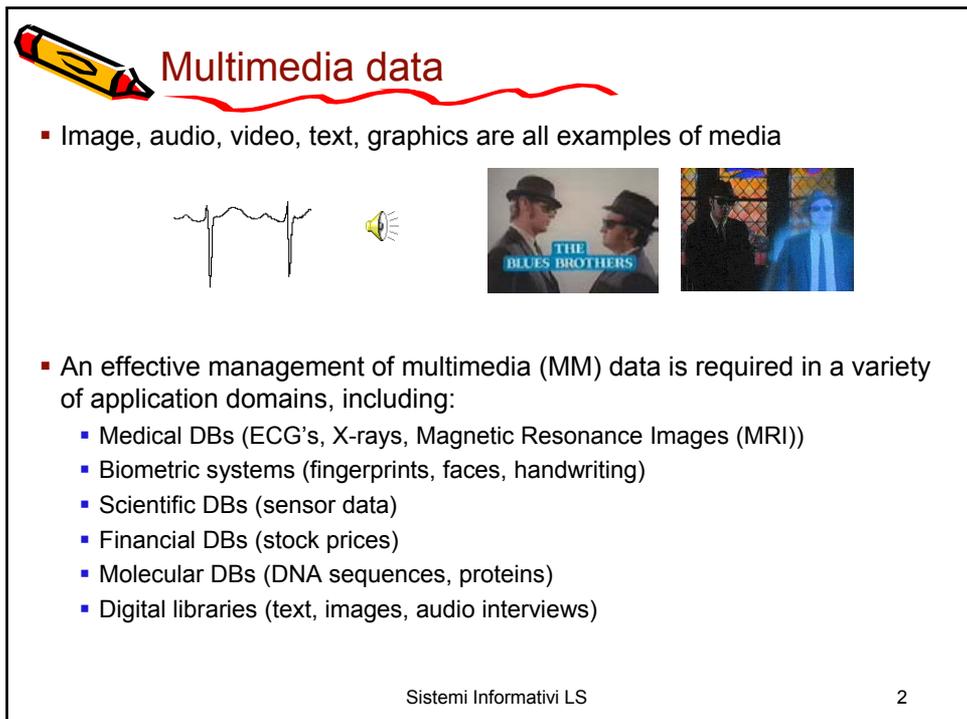


## Searching Multimedia Databases

Prof. Paolo Ciaccia  
<http://www-db.deis.unibo.it/courses/SI-LS/>

06\_SearchingMMDBs.pdf

Sistemi Informativi LS



## Multimedia data

- Image, audio, video, text, graphics are all examples of media



- An effective management of multimedia (MM) data is required in a variety of application domains, including:
  - Medical DBs (ECG's, X-rays, Magnetic Resonance Images (MRI))
  - Biometric systems (fingerprints, faces, handwriting)
  - Scientific DBs (sensor data)
  - Financial DBs (stock prices)
  - Molecular DBs (DNA sequences, proteins)
  - Digital libraries (text, images, audio interviews)

Sistemi Informativi LS

2



## Managing MM data

- There are several issues concerning the “management” of MM data, such as:
  - Representation: formats, compression (e.g., JPEG, MPEG, WAV)
  - Storage: physical layout on disk (e.g., BLOB)
  - **Search and retrieval**
  - Generation, acquisition, transmission, delivery
- In the following we concentrate on aspects related to the search and retrieval of MM objects
- We discuss “**general**” issues, that is, they are not tailored to a specific medium
- This is possible since we adopt a “**separation of competences**”:
  1. The “domain expert” knows how to search
  2. We consider this “search requirement” as given and study “general” solutions to the search problem(s)
- On point 1. we just provide illustrative examples (this is not a course on image analysis and digital signal processing!)



## Content-based search

- Searching for MM objects can rely on standard **text-based techniques**, provided objects come with a precise textual description of what they represent/describe, i.e., of their semantics
- However, the “**annotation**” of MM objects is a **subjective**, **difficult**, and **tedious** process; further it is **hardly applicable in the general case**
- A more appealing approach, suitable to manage large DBs, is to

**automatically extract from MM objects  
a set of (low-level) relevant numerical *features***

that, at least partially, convey some of the semantics of the objects

- Clearly, which are the “**best**” features to extract depend on the **specific medium and on the application at hand** (i.e., what we are looking for)



**Look for cheetahs?**



**This is fine; but, how to find it?**



## Content-based similarity search

- Once we have feature values, we can search objects by using them
- Assume we a DB with N MM objects (e.g., images) and, for each of the N objects, we have extracted the “relevant features”
  - E.g., we could extract some color information from images
- We can now search for **objects whose feature values are “similar”** (in some sense to be defined) **to the feature values of our query**
- In general, **this approach**, much alike as it happens in text-retrieval, **cannot guarantee that all and only relevant results are returned as result of a query**



Look for cheetahs?

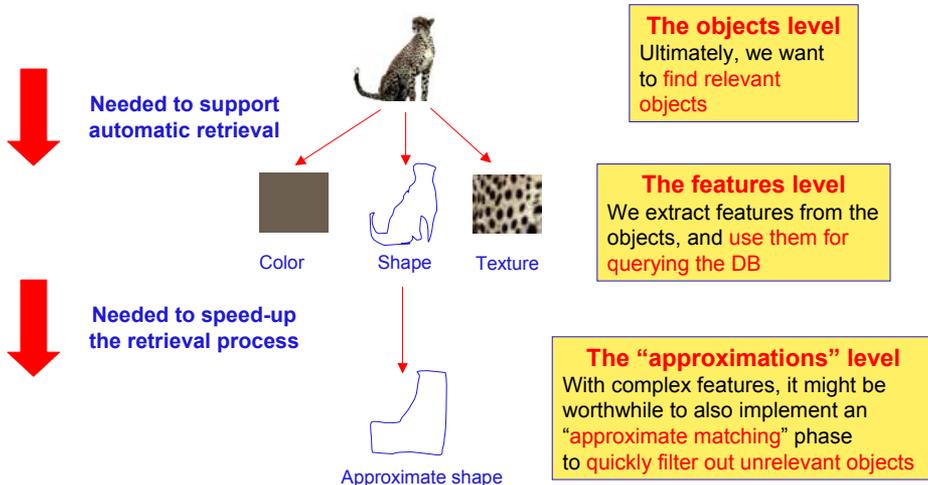


Oops! Not really a cheetah ;-)



## The general scenario

- In general, we have a 3-levels scenario:





## Approximating features: a 1<sup>st</sup> example (1)

- Consider the case where:
  - feature values are **D-dimensional vectors**, i.e.,  $p = (p_1, \dots, p_D)$
  - the (dis-)similarity of two feature values is measured using the Euclidean distance,  $L_2(p, q)$
  - we want to process **range queries**, thus looking for all p's s.t.  $L_2(p, q) \leq r$  holds
- For large values of D, e.g.,  $D \approx 100 \div 1000$ , which is **typical of MM features**, even computing a single distance value might require a non-negligible time ( $O(D)$  floating point operations)
- Rather than **directly** using  $L_2$ , we **pre-compute for each feature vector its average**:  $\bar{p} = \sum_{j=1, D} p_j / D$  and **do the same, at run-time, for the query vector**
- Then, we issue the (1-dimensional) range query  $L_1(\bar{p}, \bar{q}) = |\bar{p} - \bar{q}| \leq r/\sqrt{D}$
- All the feature vectors whose average does not satisfy the inequality can be immediately pruned away**, the others need to be checked against the actual search predicate  $L_2(p, q) \leq r$



## Approximating features: a 1<sup>st</sup> example (2)

- In order to prove the correctness of the approach we just need to show that:
 
$$L_2(p, q) \leq r \Rightarrow |\bar{p} - \bar{q}| \leq r/\sqrt{D}$$
- In other terms, our “approximate filter” should not discard any vector that satisfies the query predicate

**Proof:** We show that it is:  $\sqrt{D} \times |\bar{p} - \bar{q}| \leq \sqrt{\sum_{j=1, D} (p_j - q_j)^2}$  from which the result follows

- Squaring both sides:  $D \times |\bar{p} - \bar{q}|^2 \leq \sum_{j=1, D} (p_j - q_j)^2$
- By defining  $x_j = p_j - q_j$ :  $D \times \bar{x}^2 \leq \sum_{j=1, D} x_j^2$
- Since  $\sum_{j=1, D} (x_j - \bar{x})^2 = \sum_{j=1, D} x_j^2 + D \times \bar{x}^2 - 2 \times \sum_{j=1, D} x_j \times \bar{x} = \sum_{j=1, D} x_j^2 - D \times \bar{x}^2 \geq 0$

the result follows ■



## Approximating features: a 2<sup>nd</sup> example (1)

- Consider the case where you are managing **geometric objects with a not null spatial extension**, such as lines, polygons, etc.
- For any two objects  $o_1$  and  $o_2$  define their  $L_2$  distance as:  

$$L_2(o_1, o_2) = \min\{L_2(p_1, p_2) : p_1 \in o_1, p_2 \in o_2\}$$
- For complex geometries, computing the distance between any two objects is a computationally-demanding task
- For queries like  $L_2(o, q) \leq r$  we can proceed as follows:
  - For each object  $o$ , **pre-compute the MBR of  $o$**
  - At query time, **do the same for the query region  $q$**
- Then, issue the range query  $L_2(\text{MBR}(o), \text{MBR}(q)) \leq r$
- All the regions that do not satisfy the inequality can be immediately pruned away, the others need to be checked against the actual search predicate  $L_2(o, q) \leq r$**



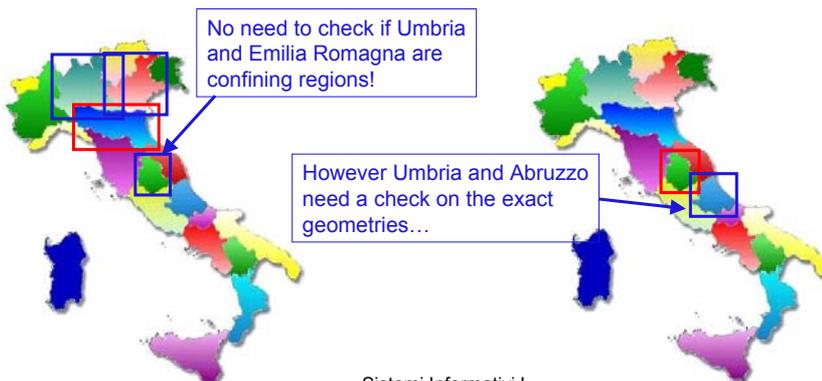
## Approximating features: a 2<sup>nd</sup> example (2)

- The method works since **MBR's are a conservative approximation of the geometries**. This is also to say that:

$$L_2(\text{MBR}(o_1), \text{MBR}(o_2)) \leq L_2(o_1, o_2)$$

from which we have

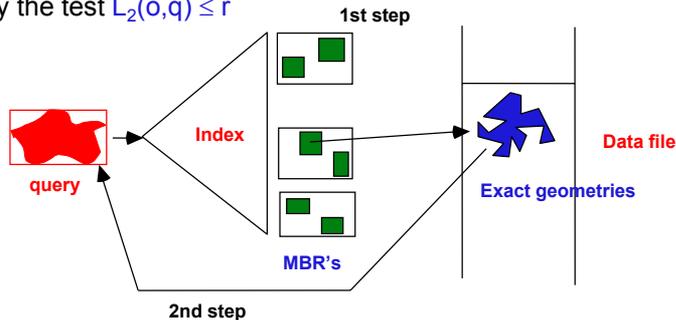
$$L_2(o_1, o_2) \leq r \Rightarrow L_2(\text{MBR}(o_1), \text{MBR}(o_2)) \leq r$$





## Approximating features: a 2<sup>nd</sup> example (3)

- From the implementation point of view, we can resort to a multi-dimensional access method, like the R-tree, to **index the MBR's of regions**
  - This is a straightforward extension of what we have already seen, in that **leaf pages now store MBR's rather than points**
- In the **first step**, we use the index and retrieve the MBR's that satisfy the predicate  $L_2(\text{MBR}(o), \text{MBR}(q)) \leq r$
- In the **second step**, we access the actual geometries of the objects and apply the test  $L_2(o, q) \leq r$



Sistemi Informativi LS

11



## The Filter & Refine strategy

- The previous examples are just particular cases of a more general approach, which can be adopted whenever directly working at the feature level is computationally expensive; the approach is termed **Filter & Refine** since we first apply a (quick and approximate) filter test, and then we refine the result
- There are two basic requirements:
  1. The filter phase must not discard any qualifying object
    - No “false dismissals”
  2. The number of “candidate” objects that survive the filter step should not be too high, as compared to the number of actually qualifying objects
    - Not too many “false drops”
- Note that 1. concerns the **correctness** of the approach, whereas 2. concerns its **efficiency**
- These notions of “false dismissals” and “false drops” are somewhat different from the ones we have discussed in the context of IR systems, since **here no user evaluation (relevance) is concerned**

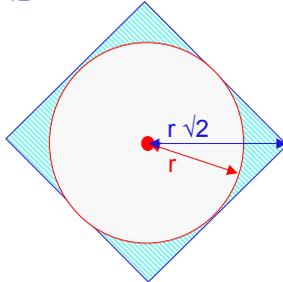
Sistemi Informativi LS

12



## Filter & Refine without feature approximation

- A particular case of the Filter & Refine approach arises when the filter step works on the original feature values, i.e., no feature approximation is computed
- In this case, the speed-up due to the filter step is obtained by **replacing the original distance function with one simpler to compute**
- As a simple example, for the query  $L_2(p,q) \leq r$  one could consider to save  $D$  floating-point operations by using the  $L_1$  distance function and the query  $L_1(p,q) \leq r \times \sqrt{D}$
- Graphically:



Sistemi Informativi LS

13



## The Lower-bounding Lemma

- The so-called **lower-bounding (L-B) lemma** is a basic tool, which summarizes the specific cases seen so far, to deal with similarity queries over **complex feature domains and/or complex distance functions**
- It has been successfully applied in a large variety of cases, including retrieval of color images, tumor-like shapes, time series, geometric DB's, DNA and protein DB's, etc.

### Lower-bounding Lemma:

- Given a range query  $\{p:d(p,q) \leq r\}$ , and a filter step that uses the distance function  $\delta$  on the, possibly approximate, feature values  $p'$  and  $q'$ , in order to guarantee the absence of false dismissals it is sufficient to have:

$$\delta(p',q') \leq d(p,q) \quad \forall p',q',p,q$$

**Proof:** Immediate, since  $d(p,q) \leq r$  implies  $\delta(p',q') \leq r$  ■

- Intuitively, in the filter step we just require that objects appear “closer” than they actually are

Sistemi Informativi LS

14



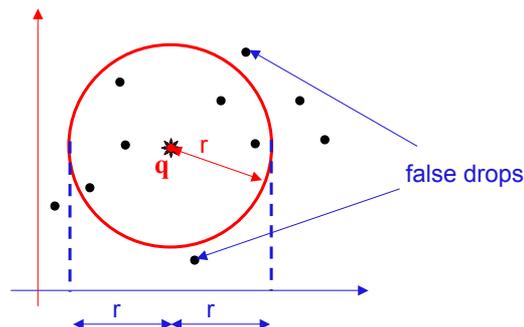
## The L-B Lemma and dim. reduction

- A common approach to approximate feature values is to perform a so-called *dimensionality reduction*
- Besides reducing the cost of computing distances, **this is sometimes necessary to achieve efficient index performance**, since **in high-dimensional spaces the performance of index structures degenerates**
  - This is known as the “**dimensionality curse**” problem
- The basic idea of dimensionality reduction is to transform  $D$ -dimensional vectors into vectors of a space at lower dimensionality,  $D' \ll D$
- This is sometimes achieved by “**projecting**” vectors by means of a **linear transformation**, as specified by a  $D' \times D$  matrix, thus **the new coordinates are linear combinations of the original ones**
  - E.g., the “average” approximation follows this pattern
- When approximation is obtained this way, for the L-B lemma to apply we need to guarantee that the linear transformation is a “contractive” one (i.e., distances do not increase)



## Dimensionality reduction: a trivial example

- A trivial way to perform dimensionality reduction is simply to discard some of the features' coordinates
- This works since distance functions on vector spaces, say  $L_p$ -norms, are non-decreasing if we add more coordinates





## k-NN queries and the L-B lemma

- The L-B lemma is for range queries; what about k-NN queries? In particular:
  - if we have an index built on the approximate feature values,  $p'$ , and an approximate distance function  $\delta$ ,
  - can we use them to find the k-NN's of a query point  $q$  according to the distance function  $d$ ?

**YES!**

- We see two algorithms, the first [KFS+96] quite simple and making use of range and k-NN queries, the second [SK98] optimal and based on distance browsing



## A 1st algorithm for multi-step k-NN queries

- The [KFS+96] algorithm works in 4 steps:
  1. Execute a k-NN query  $NN(q', \delta, k)$  using the index.  
Let  $Resk\text{-}NN$  be the result set of this step
  2. For each  $p' \in Resk\text{-}NN$  compute its actual distance  $d(p, q)$  from  $q$ .  
Let  $r = \max\{d(p, q) : p' \in Resk\text{-}NN\}$  be the maximum of such distances
  3. Execute a range query  $Range(q', \delta, r)$  using the index, thus retrieving all (approximate) feature values such that  $\delta(p', q') \leq r$ .  
Let  $ResRange$  be the result set of this step
  4. For each  $p' \in ResRange$  compute its actual distance  $d(p, q)$  from  $q$ , and return the k objects with minimum distance



## The [KFS+96] algorithm: an example

Let  $k=2$

|                 | p1 | p2 | p3 | p4 | p5 | p6 | p7 | p8 | p9 | p10 | p11 |
|-----------------|----|----|----|----|----|----|----|----|----|-----|-----|
| $d(p,q)$        | 15 | 12 | 5  | 3  | 8  | 10 | 13 | 9  | 10 | 14  | 12  |
| $\delta(p',q')$ | 10 | 7  | 4  | 2  | 3  | 9  | 10 | 7  | 9  | 12  | 9   |

- In the 1st step we retrieve  $\text{Resk-NN} = \{p4, p5\}$
- In the 2nd step we set  $r = \max\{3, 8\} = 8$
- In the 3rd step we retrieve  $\text{ResRange} = \{p2, p3, p4, p5, p8\}$
- In the 4th step we determine the final result, i.e.,  $p3$  and  $p4$

### The [KFS+96] algorithm is correct

Proof:

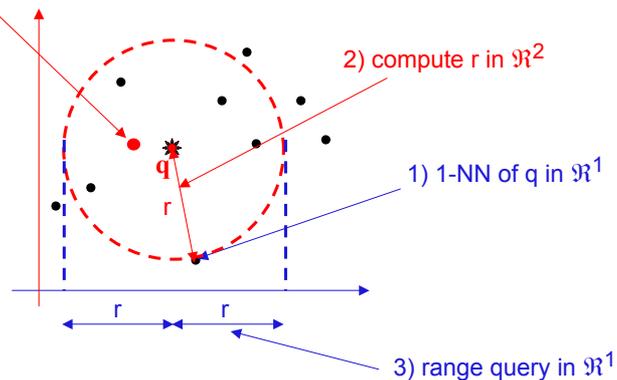
- It suffices to show that, if  $p^*$  is one of the  $k$ -NN's of  $q$ , then the corresponding  $p^* \in \text{ResRange}$ .
- Assume this is not the case, i.e.,  $\delta(p^*, q') > r = \max\{d(q, p) : p' \in \text{Resk-NN}\}$
- Since  $d(p^*, q) \geq \delta(p^*, q')$  holds (L-B lemma), this would imply that there are  $k$  points (those in  $\text{Resk-NN}$ ) whose distance from  $q$  is less than that of  $p^*$ , thus contradicting the hypothesis that  $p^*$  is a  $k$ -NN of  $q$  ■



## A graphical example

- Consider the case when feature values are projected from  $\mathbb{R}^2$  to  $\mathbb{R}^1$  and  $k=1$

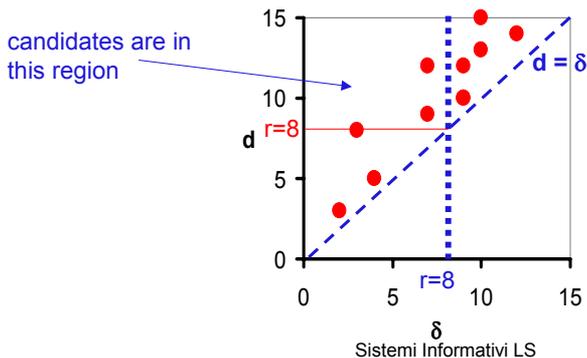
4) Determine 1-NN of  $q$  in  $\mathbb{R}^2$





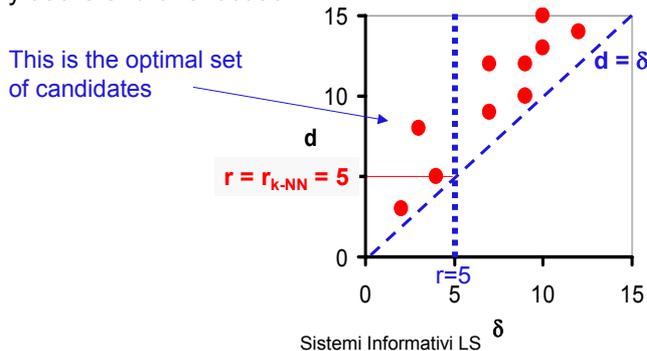
## Limits of the [KFS+96] algorithm

- Although correct, we can argue that the [KFS+96] algorithm unnecessarily generates **too many candidates**
- We can visualize what's going on by representing the indexed points in a  $(\delta, d)$  distance space
  - For each point  $p$ , its coordinates are the values of  $\delta(p', q')$  and  $d(p, q)$ , respectively



## The [SK98] intuition

- The basic intuition of the [SK98] algorithm is that it is possible to drop from the set of candidates those **points  $p'$  such that  $\delta(p', q') > r_{k-NN}$** , where  $r_{k-NN}$  is the actual distance of the  $k$ -th NN of  $q$ 
  - This holds because of the L-B lemma:  $d(p, q) \geq \delta(p', q') > r_{k-NN}$
- What remains is the **"optimal set of candidates"**
- However, as with the  $k$ NNOptimal algorithm, we know the value of  $r_{k-NN}$  only at the end of execution...





## An optimal multi-step k-NN algorithm

- The [SK98] algorithm works as follows:

### 1. Initialize

- Make  $k$  distance browsing calls to the index (next-NN algorithm)
- For each returned point  $p'$  compute its actual distance  $d(p,q)$  from  $q$  and store  $(p,d(p,q))$  pairs in a ResultList
- Let  $r_{k-NN} = \max_p \{d(p,q)\}$  be the maximum of such distances (=ResultList(k).dist)

### 2. Repeat

- Retrieve from the index the next NN according to  $\delta$ .  
Let its distance be  $\rho = \delta(p',q')$
- If  $\rho < r_{k-NN}$  compute  $d(p,q)$  and possibly update the ResultList and  $r_{k-NN}$

### 3. Until $\rho > r_{k-NN}$

### 4. Return ResultList



## The [SK98] algorithm: an example

Let  $k=2$

|                 | p1 | p2 | p3 | p4 | p5 | p6 | p7 | p8 | p9 | p10 | p11 |
|-----------------|----|----|----|----|----|----|----|----|----|-----|-----|
| $d(p,q)$        | 15 | 12 | 5  | 3  | 8  | 10 | 13 | 9  | 10 | 14  | 12  |
| $\delta(p',q')$ | 10 | 7  | 4  | 2  | 3  | 9  | 10 | 7  | 9  | 12  | 9   |

- We initially have ResultList =  $\{(p4,3),(p5,8)\}$  and  $r_{k-NN} = 8$
- The next call to the index returns  $p3$ , for which it is  $\rho = \delta(p3,q') = 4$  and  $d(p3,q) = 5$
- This leads to ResultList =  $\{(p4,3),(p3,5)\}$  and  $r_{k-NN} = 5$
- Since  $\rho = 4 < r_{k-NN} = 5$  we execute another distance browsing step
- This returns either  $p2$  or  $p8$ , for both of which it is  $\rho = 7$
- Since now it is  $\rho = 7 > r_{k-NN} = 5$  we can stop and return the ResultList



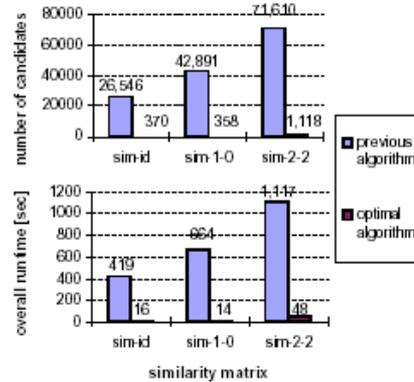
## Some results (from [SK98] presentation) (1)

- Uniformly distributed data (synthetic dataset)

- Experimental Setting**
- 100,000 Objects, **20-D** (L2, Weighted L2)
  - Matrices: **sim-id, 1-0, 2-2**
  - Queries:  $k = 10$  (0.01%)
  - Index: 15-D**

### Avg. Improvement Factors

- Candidates: 72, 120, 64
- Overall Time: 26, 48, 23



(c) 1998 Thomas Seidl

SIGMOD '98 - 12



## Some results (from [SK98] presentation) (2)

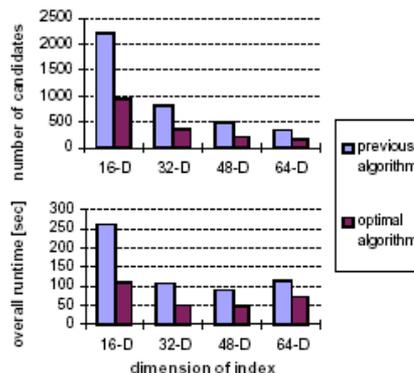
- Similarity of objects' shapes ( $D = 1024$ )
- Evaluating a single  $d(p,q)$  requires about 100 msecs
  - With 10,000 objects, a sequential evaluation would take about 1,000 seconds plus the I/O time!

### Experimental Setting

- 10,000 Images, 32x32 Pixel
- 'Neighborhood Area': 9-1
- Queries:  $k = 5$  (0.05%)
- Index (KLT): 16-D, ..., 64-D**

### Avg. Improvement Factors

- Candidates: 2.3
- Overall Time: 1.6 to 2.3



(c) 1998 Thomas Seidl

SIGMOD '98 - 13



## Some results (from [SK98] presentation) (3)

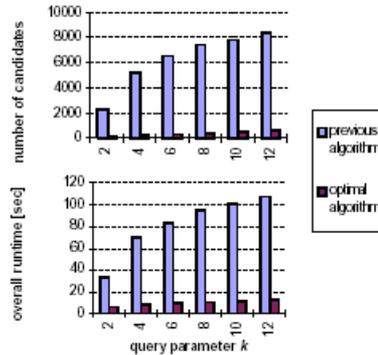
- Silimarity of images based on color histograms ( $D = 112$ )
- Evaluating a single  $d(p,q)$  requires about 1.1 msec
  - With 100,000 objects, a sequential evaluation would take about 100 seconds plus the I/O time!

### Experimental Setting

- 112,700 Histograms (112-D)
- Quadratic Form Distance
- Queries:  $k = 2, \dots, 12$  (0.01%)
- Index (KLT): 12-D

### Avg. Improvement Factors

- Candidates: 17
- Overall Time: 8.5



(c) 1998 Thomas Seidl

SIGMOD '98 - 14

Sistemi Informativi LS

27



## Some considerations

- From a **DB point of view**, most of the work concerned with searching in MMDB's reduces to

finding smart lower-bounding techniques that make it possible to apply a filter & refine evaluation strategy

- Clearly, since features are media-dependent, there is no "universal" solution, that is:

each feature type and each (dis-)similarity criterion bring a new specific problem to be solved

- We will just look at a few relevant examples, many others are omitted for time reasons, but they can be easily dealt with once one has captured the basic idea
- In order to effectively implement the "*separation of competences*" approach

it is useful to rely on powerful abstractions that allow us to apply our solutions/algorithms to a variety of, apparently unrelated, problems

- Relevant case: **metric indexes**, which work on any **metric space**

Sistemi Informativi LS

28