

## Raw Images

---

The content of an image consists of all “interesting” objects in that image. Each object is characterized by:

- a *shape descriptor* that describes the shape/location of the region within which the object is located inside a given image.
- a *property descriptor* that describes the properties of the individual pixels (or groups of pixels) in the given image. Examples of such properties include: red-green-blue (RGB) values of the pixel (or aggregated over a group of pixels), grayscale levels in the case of black and white images, etc. In general, it will be infeasible to associate properties with individual pixels, and hence, cells (rectangular “groups” of pixels) will be used most of the time.
- We assume the existence of a set **Prop** of properties. A property consists of two components –
  1. A *property name* – e.g. “Red”, “Green”, “Blue” and
  2. a *property domain* which specifies the range of values that the property can assume – e.g.  $\{0, \dots, 8\}$ .

## Example

---

**Example:** Consider the image file `pic1.gif` on the preceding slide. This image has two objects of interest - let call these two objects  $o_1$  and  $o_2$ .

- The shapes of these objects are captured by rectangles shown. Formally, object  $o_1$ 's shape may be specified by:

*rectangle* :  $XLB = 10; XUB = 60, YLB = 5; YUB = 50$ .

- The *property descriptor* associated with an individual cell (group of pixels) may look like this:

1. Red = 5;
2. Green = 1;
3. Blue = 3.

Other properties like texture may also be included.

## Definitions

---

- Every image  $I$  has an associated pair of positive integers  $(m, n)$ , called the *grid-resolution* of the image. This divides the image into  $(m \times n)$  *cells* of equal size, called the *image grid*.
- Each cell in a given gridded  $(m \times n)$  image  $I$  consists of a collection of pixels.
- A *cell property* is a triple (**Name**, **Values**, **Method**) where **Name** is a string denoting the property's name, **Values** is a set of values that that property may assume, and **Method** is an algorithm that tells us how to compute the property involved.

**Example:** Consider black and white images.

$(\text{bwcolor}, \{\text{b}, \text{w}\}, \text{bwalgo})$

could be a cell property with property name **bwcolor** and the possible values are **b** (black) and **w** (white), respectively. **bwalgo** then is an algorithm which may take a cell as input, and return as output, either black or white, by somehow combining the black/white levels of the pixels in the cell.

**Example:** Consider gray scale images (with 0 =white and 1 =black), we may have the cell property

$(\text{graylevel}, [0, 1], \text{grayalgo})$

where our property name is named **graylevel**, and its possible values are real numbers in the  $[0, 1]$  interval, and the associated method **grayalgo** takes as input, a cell, and computes its gray level.

## Image Definitions

---

- An *object shape* is any set  $P$  of points such that if  $p, q \in P$ , then there exists a sequence of points  $p_1, \dots, p_n$  all in  $P$  such that:

1.  $p = p_1$  and  $q = p_n$  and
2. for all  $1 \leq i < n$ ,  $p_{i+1}$  is a neighbor of  $p_i$ , i.e. if  $p_i = (x_i, y_i)$  and  $p_{i+1} = (x_{i+1}, y_{i+1})$ , then  $(x_{i+1}, y_{i+1})$  satisfies one of the following conditions:

$$\begin{array}{ll}
 (x_{i+1}, y_{i+1}) = (x_i + 1, y_i) & (x_{i+1}, y_{i+1}) = (x_i - 1, y_i) \\
 (x_{i+1}, y_{i+1}) = (x_i, y_i + 1) & (x_{i+1}, y_{i+1}) = (x_i, y_i - 1) \\
 (x_{i+1}, y_{i+1}) = (x_i + 1, y_i + 1) & (x_{i+1}, y_{i+1}) = (x_i + 1, y_i - 1) \\
 (x_{i+1}, y_{i+1}) = (x_i - 1, y_i + 1) & (x_{i+1}, y_{i+1}) = (x_i - 1, y_i - 1)
 \end{array}$$

- A rectangle is an object shape,  $P$ , such that there exist integers  $XLB, XUB, YLB, YUB$  such that

$$P = \{(x, y) \mid XLB \leq x < XUB \& YLB \leq y < YUB\}.$$

## Image Database

---

**Def:** An *image database*, **IDB**, consists of a triple (**GI**, **Prop**, **Rec**) where:

1. **GI** is a set of gridded images of the form  $(Image, m, n)$  and
2. **Prop** is a set of cell properties, and
3. **Rec** is a mapping that associates with each image, a set of rectangles denoting objects.

## Issues in Image Databases

---

- First and foremost, images are often very large objects consisting of a  $(p_1 \times p_2)$  pixel array. Explicitly storing properties on a pixel by pixel basis is usually infeasible. This has led to a family of *image compression* algorithms that attempt to compress the image into one containing fewer pixels.
- Given an image  $I$  (compressed or raw), there is a critical need to determine what “features” appear in the image. This is typically done by breaking up the image into a set of homogeneous (w.r.t. some property) rectangular regions, each of which is called a *segment*. The process of finding these segments is called *segmentation*.
- Once image data has been segmented, we need to support “match” operations that map either a whole image or a segmented portion of an image against another whole/segmented image.

## Compressed Image Representations

---

- Consider a 2-dimensional image  $I$  consisting of  $(p_1 \times p_2)$  pixels.
- Let  $I(x, y)$  be a number denoting one or more attributes of the pixel.
- The creation of the compressed representation,  $\text{cr}(I)$ , of image  $I$  consists of two parts:
  1. **Size Selection:** The larger the size, the greater is the fidelity of the representation. However, as the size increases, so does the complexity of creating an index for manipulating such representations, and searching this index. Let  $\text{cr}(I)$  be of size  $h_1 \times h_2$  where  $h_i \leq p_i$ .
  2. **Transform Selection:** The user must select a transformation, which, given the image  $I$ , and any pairs of number  $1 \leq i \leq h_1$ , and  $1 \leq j \leq h_2$  will determine what the value of  $\text{cr}(i, j)$  is.
  3. There are many such transforms.



## The Discrete Fourier Transform (DFT)

---

$$\underline{\text{DFT}}(i, j) = \frac{1}{\sqrt{p_1}} \times \frac{1}{\sqrt{p_2}} \times \sum_{a=0}^{p_1} \sum_{b=0}^{p_2} \left( I(a, b) \times \underline{\exp} \left( \frac{-2\pi \underline{j} a \times i}{p_1} \right) \times \frac{-2\pi \underline{j} b \times i}{p_2} \right).$$

where:  $j$  is the well known complex number,  $\sqrt{-1}$ .

DFT has many nice properties.

- **Invertibility:** It is possible to “get back” the original image  $I$  from its DFT representation. Useful for decompression.
- *Note that practical realizations of DFT ave often sacrificed this property by applying the DFT together with certain other non-invertible operations.*
- **Distance Preservation:** DFT preserves Euclidean distance. This is important in image matching applications where we may wish to use distance measures to represent similarity levels.

## The Discrete Cosine Transform

---

$$\text{DCT}(i, j) = \frac{2}{\sqrt{p_1 \times p_2}} \alpha(i) \times \alpha(j) \sum_{r=0}^{p_1-1} \sum_{s=0}^{p_2-1} \left( \cos \left( \frac{(2r+1) \times \pi i}{2r} \right) \times \cos \left( \frac{(2s+1) \times \pi j}{2s} \right) \right)$$

where:

$$\alpha(i), \alpha(j) = \begin{cases} \frac{1}{\sqrt{2}} & \text{when } u, v = 0 \\ 1 & \text{otherwise.} \end{cases}$$

- DCT also is easily invertible
- DCT can be computed quite fast.

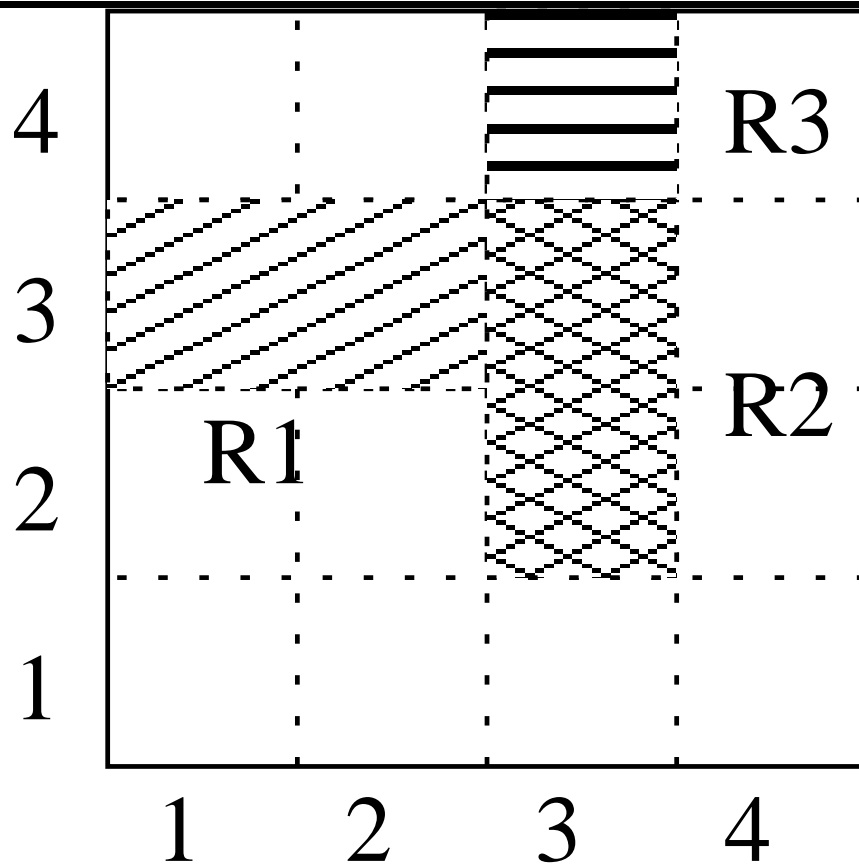
Other compression techniques include Wavelets.

## Image Processing: Segmentation

---

- This is the process of taking as input, an image, and producing as output, a way of “cutting up” the image into disjoint regions such that each region is “homogeneous”.
- Suppose  $I$  is an image containing  $(m \times n)$  cells.
- A *connected region*,  $\mathfrak{R}$ , in image  $I$ , is a set of cells such that if cells  $(x_1, y_1), (x_2, y_2) \in \mathfrak{R}$ , there there exists a sequence of cells  $C_1, \dots, C_n$  in  $\mathfrak{R}$  such that:
  1.  $C_1 = (x_1, y_1)$  and
  2.  $C_n = (x_2, y_2)$  and
  3. The Euclidean distance between cells  $C_i$  and  $C_{i+1}$  for all  $i < n$  is 1.

## Example: Connected Regions



- Each of  $R_1$ ,  $R_2$ ,  $R_3$  is a connected region.
- $(R_1 \cup R_2)$  is a connected region;
- $(R_2 \cup R_3)$  is a connected region;
- $(R_1 \cup R_2 \cup R_3)$  is a connected region;
- But  $(R_1 \cup R_3)$  is *not* a connected region. The reason for this is that the Euclidean distance between the cell  $(2, 3)$  which represents the rightmost of the two cells of  $R_1$  and the cell  $(3, 4)$  which represents the only cell of  $R_3$  is  $\sqrt{2} > 1$ .

## Homogeneity Predicates

---

- A *homogeneity predicate* associated with an image  $I$  is a function  $H$  that takes as input, any connected region  $\mathfrak{R}$  in image  $I$ , and returns either “true” or “false.”
- **Example:** Suppose  $\delta$  is some real number between 0 and 1, inclusive, and we are considering black and white images. We may define a simple homogeneity predicate,  $H_\delta^{bw}$  as follows:  $H_\delta^{bw}(R)$  returns “true” if over  $(100 * \delta)\%$  of the cells in region  $R$  have the same color.

- Consider three regions now, as described in the following table:

Region	Num. of Black Pixels	Num. of White Pixels
$R_1$	800	200
$R_2$	900	100
$R_3$	100	900

- Suppose we consider some different predicates,  $H_{0.8}^{bw}$ ,  $H_{0.89}^{bw}$  and  $H_{0.92}^{bw}$ .
- The following table shows us the results returned by these three homogeneity predicates on the above table  $R$ .

Region	$H_{0.8}^{bw}$	$H_{0.89}^{bw}$	$H_{0.92}^{bw}$
$R_1$	true	false	false
$R_2$	true	true	false
$R_3$	true	true	false

## Another Homogeneity Predicate Example

---

- Suppose each pixel has a real value between 0 and 1, inclusive.
- This value is called the bw-level.
- 0 denotes “white”, 1 denotes “black”, and everything in between denotes a shade somewhere between black and white.
- Suppose  $f$  assigns numbers between 0 and 1 (inclusive) to each cell. In addition, you have a “noise factor”  $0 \leq \eta \leq 1$ , and a threshold  $\delta$  as in the preceding case.
- $H^{f,\eta,\delta}(R)$  is now “true” iff

$$\frac{\{(x, y) \mid |\text{bwlevel}(x, y) - f(x, y)| < \eta\}}{(m \times n)} > \delta.$$

- What this homogeneity predicate does is to use a “baseline” function  $f$ , and a maximal permissible noise level  $\eta$ . It considers the bw-level of cell  $(x, y)$  to be sufficiently similar to that predicted by  $f$  if

$$|\text{bwlevel}(x, y) - f(x, y)| < \eta,$$

i.e. if the two differ by no more than  $\eta$ .

- It then checks to see if sufficiently many cells (which is determined by the factor  $\delta$ ) in the region “match” the predictions made by  $f$ . If so, it considers the region  $R$  to be homogeneous, and returns “true.” Otherwise, it returns “false.”

## Segmentation

---

**Def:** Given an image  $I$  represented as a set of  $(m \times n)$  pixels, we define a *segmentation* of image  $I$  w.r.t. a homogeneity predicate  $P$  to be a set  $R_1, \dots, R_k$  of regions such that:

1.  $R_i \cap R_j = \emptyset$  for all  $1 \leq i \neq j \leq k$  and
2.  $I = R_1 \cup \dots \cup R_k$ ;
3.  $H(R_i) = \text{"true"}$  for all  $1 \leq i \leq k$ ;
4. For all distinct  $i, j$ ,  $1 \leq i, j \leq n$  such that  $R_i \cup R_j$  is a connected region, it is the case that  $H(R_i \cup R_j) = \text{"false."}$

**Example:** For example, consider a simple  $(4 \times 4)$  region containing the bw-levels shown in the table below.

Row/Col	1	2	3	4
1	0.1	0.25	0.5	0.5
2	0.05	0.30	0.6	0.6
3	0.35	0.30	0.55	0.8
4	0.6	0.63	0.85	0.90

Consider now, the homogeneity predicate  $H_1^{dyn,0.03}$ . This homogeneity predicate says that a region  $R$  is to be considered homogeneous iff there exists an  $r$  such that each and every cell in the

region has a bw-level  $v$  such that

$$|v - r| \leq 0.03.$$

According to this classification, it is easy to see that the following five regions constitute a valid segmentation of the above image w.r.t.  $H_1^{dyn,0.03}$ .

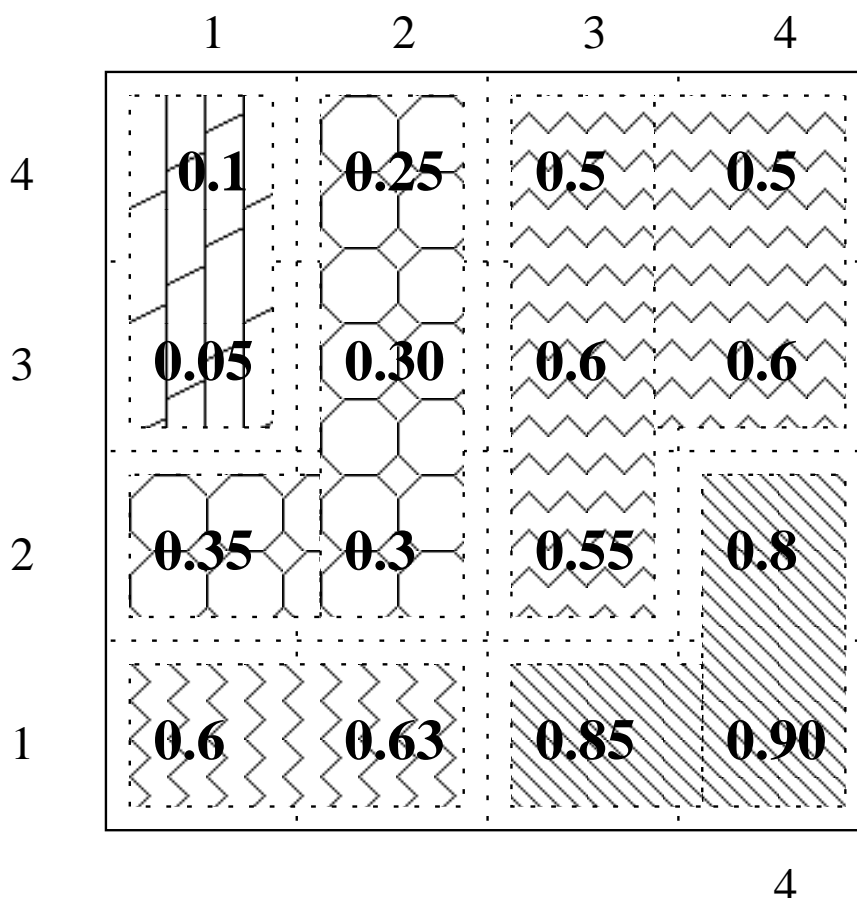
$$R_1 = \{(1, 1), (1, 2)\}.$$

$$R_2 = \{(1, 3), (2, 1), (2, 2), (2, 3)\}.$$

$$R_3 = \{(3, 1), (3, 2), (3, 3), (4, 1), (4, 2)\}.$$

$$R_4 = \{(3, 4), (4, 3), (4, 4)\}.$$

$$R_5 = \{(1, 4), (2, 4)\}.$$





## Segmentation Algorithm Sketch

---

- **Split:** In this method, we start with the whole image. If it is homogeneous, then we are done, and the image is a valid segmentation of itself. Otherwise, we split the image into two parts, and recursively repeat this process, till we find a set  $R_1, \dots, R_n$  of regions that are homogeneous, and satisfy all conditions, except the fourth condition in the definition of “homogeneity” predicates.
- **Merge:** We now check which of the  $R_i$ ’s can be merged together. At the end of this step, we will obtain a valid segmentation  $R'_1, \dots, R'_k$  of the image, where  $k \leq n$  and where each  $R'_i$  is the union of some of the  $R_j$ ’s.

## Segmentation Algorithm

---

```
function segment(I:image);  
    SOL =  $\emptyset$ ;  
    check_split(I);  
    merge(SOL);  
end function  
  
function check_split(R);  
    if  $H(R) = \text{''true''}$  then addsol(R)  
    else  
        { X = split(R);  
          check_split(X.part1);  
          check_split(X.part2);  
        }  
end function  
  
procedure addsol(R);  
    SOL = SOL  $\cup$  {R}  
end procedure
```

## Segmentation Algorithm (Contd.)

---

```

function merge(S);
  while  $S \neq \emptyset$  do {
    Pick some Cand in S;
    merged = false;
     $S = S - \{Cand\}$ ;
    Enumerate S as  $C_1, \dots, C_k$ ;
    while  $i \leq k$  do
      { if adjacent(Cand,  $C_i$ ) then
        {  $Cand = Cand \cup C_i$ ;
           $S = S - \{C_i\}$ ;
          merged = true;
        }
      else {  $i = i + 1$ ;
        if merged then  $S = S \cup \{Cand\}$ ;
        merged = false;
      }
    }
  }
end function

```

## Similarity Based Retrieval

---

Which of these images is similar to the other?



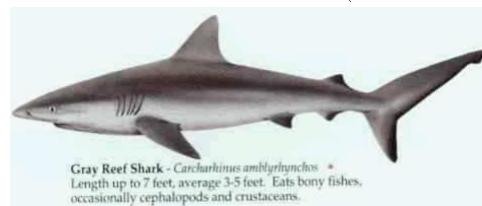
(a) One Monkey (chimp)



(b) Another Monkey (orangutan)



(a) One Shark (tiger)



(b) Another Shark (grayreef)

## Similarity Based Retrieval

---

Two approaches:

- **(The Metric Approach)** Assume there is a distance metric  $d$  that can compare any two image objects. The closer two objects are in distance, the more similar they are considered to be. *Given an input image  $i$ , find the “nearest” neighbor of  $i$  in the image archive.* This is the most widely followed approach in the database world.
- **(The Transformation Approach)** The metric approach assumes that the notion of similarity is “fixed”, i.e. in any given application only one notion of similarity is used to index the data (though different applications may use different notions of similarity). Computes the “cost” of transforming one image into another based on user-specified cost functions that may vary from one query to another.

## Metric Approach

---

- Suppose we consider a set  $Obj$  of objects, having pixel properties  $p_1, \dots, p_n$ , as described earlier in this chapter. Thus, each object  $o$  may be viewed as a set  $S(o)$
- A function  $d$  from some set  $X$  to the unit interval  $[0, 1]$  is said to be a distance function if it satisfies the following axioms for all  $x, y, z \in X$ :

$$\begin{aligned} d(x, y) &= d(y, x). \\ d(x, z) &\leq d(x, y) + d(y, z). \\ d(x, x) &= 0. \end{aligned}$$

- Let  $d_{Obj}$  be a distance function on the space of all objects in our domain, i.e.  $d_{Obj}$  is a distance function on a  $k = (n + 2)$  dimensional space.
- **Example:**  $Obj$  to consist of  $(256 \times 256)$  images having three attributes (red, green, blue) each of which assumes a value from the set  $\{0, \dots, 7\}$ . Could have:

$$\begin{aligned} d_i(o_1, o_2) &= \sqrt{\sum_{i=1}^{256} \sum_{j=1}^{256} (diff_r[i, j] + diff_g[i, j] + diff_b[i, j])} \\ diff_r[i, j] &= (o_1[i, j].red - o_2[i, j].red)^2 \\ diff_g[i, j] &= (o_1[i, j].green - o_2[i, j].green)^2 \\ diff_b[i, j] &= (o_1[i, j].blue - o_2[i, j].blue)^2 \end{aligned}$$

- Such computations can be cumbersome because the double summation leads to 65536 expressions being computed inside the sum.

## Metric Approach

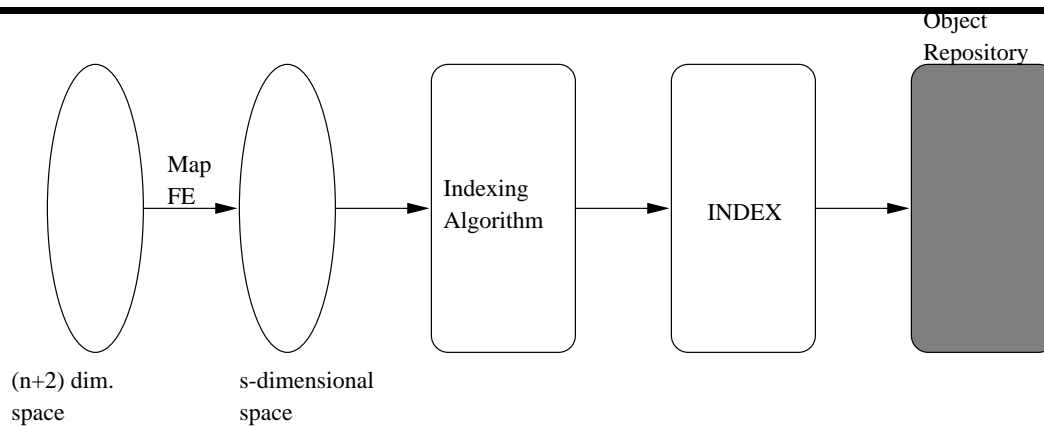
---

- How can this massive similarity computation be avoided?
- Suppose we have a “good” feature extraction function  $\underline{\mathbf{fe}}$ .
- Use  $\underline{\mathbf{fe}}$  to map objects into single points in a  $s$ -dimensional space, where  $s$  would typically be pretty small compared to  $(n + 2)$ .
- This leads to two reductions:
  1. First, recall that an object  $o$  is a *set of points* in an  $(n + 2)$  dimensional space. In contrast,  $\underline{\mathbf{fe}}(o)$  is a *single point*.
  2. Second,  $\underline{\mathbf{fe}}(o)$  is a point in an  $s$ -dimensional space where  $s \ll (n + 2)$ .
- Mapping must preserve distance, i.e. if  $o_1, o_2, o_3$  are objects such that the distance  $d(o_1, o_2) \leq d(o_1, o_3)$ , then  $d'(\underline{\mathbf{fe}}(o_1), \underline{\mathbf{fe}}(o_2)) \leq d'(\underline{\mathbf{fe}}(o_1), \underline{\mathbf{fe}}(o_3))$  where  $d$  is a metric on the original  $(n + 2)$  dimensional space, and  $d'$  is a metric on the new,  $s$ -dimensional space. In other words, the feature extraction map should preserve the distance relationships in the original space.



## Reducing Dimensionality of Feature Space

---



## Index Creation Algorithm

---

Input:  $Obj$ , a set of objects.

1.  $T = NIL$ . (\*  $T$  is an empty quadtree, or R-tree for  $s$ -dimensional data \*)
2. **if**  $Obj = \emptyset$  **then return**  $T$  **and halt**.
3. **else**
  - (a) Compute  $\underline{fe}(o)$ ,
  - (b) Insert  $\underline{fe}(o)$  into  $T$ .
  - (c)  $Obj = Obj - \{o\}$ .
  - (d) Goto 2

## Finding the Best Matches

---

### FindMostSimilarObject Algorithm

Input: a tree  $T$  of the above type. An object  $o$ .

1.  $\text{bestnode} = \text{NIL}$ ;
2. **if**  $T = \text{NIL}$  **then return**  $\text{bestnode}$ . **Halt**
3. **else**
  - find the nearest neighbors of  $\underline{\text{fe}}(o)$  in  $T$  using a nearest neighbor search technique. If multiple such neighbors exist, return them all.

## Finding “Sufficiently” Similar Objects

---

### FindSimilarObjects Algorithm

Input: a tree  $T$  of the above type. An object  $o$ . A tolerance  $0 < \epsilon \leq 1$ .

1. Execute a range query on tree  $T$  with center  $\underline{\mathbf{fe}}(o)$  and radius  $\epsilon$ .
2. Let  $o_1, \dots, p_r$  be all the points returned.
3. **for**  $i = 1$  **to**  $r$  **do**
  - (a) if  $d(o, \underline{\mathbf{fe}}^{-1}(o_i)) \leq \epsilon$  **then print**  $\underline{\mathbf{fe}}^{-1}(o_i)$ .

The above algorithm works *only if* the distance metric in the space of small dimensionality (i.e. dimension  $s$ ) *consistently overestimates* the distance metric  $d$ .

## The Transformation Approach

---

- Based on the principle that given two objects  $o_1, o_2$ , the level of *dis-similarity* between  $o_1, o_2$  is proportional to the (minimum) cost of transforming object  $o_1$  into object  $o_2$ , or vice-versa.
- We start with a set of *transformation operators*,  $to_1, \dots, to_r$ , e.g.
  - translation
  - rotation
  - scaling – uniform and nonuniform
  - excision (that culls out a part of an image)
- The transformation of object  $o$  into object  $o'$  is a *sequence* of transformation operations  $to_1, \dots, to_r$  and a sequence of objects  $o_1, \dots, o_r$  such that:
  1.  $to_1(o) = o_1$  and
  2.  $to_i(o_{i-1}) = o_i$  and
  3.  $to_r(o_r) = o'$ .

The *cost* of the above transformation sequence,  $TS$  is given by:

$$\text{cost}(TS) = \sum_{i=1}^r \text{cost}(to_i).$$

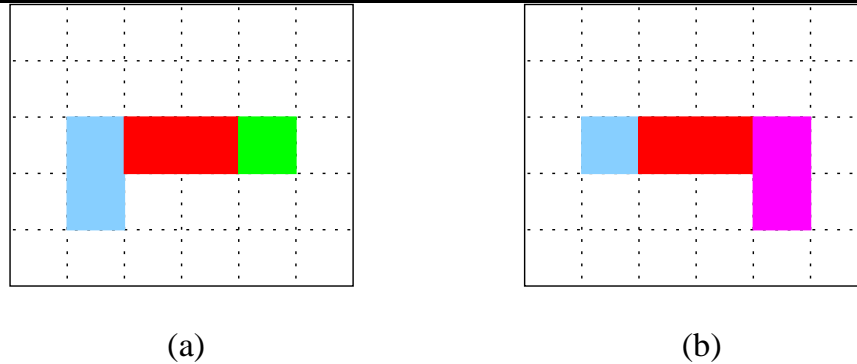
## The Transformation Approach

---

- Suppose  $\mathbf{TSeq}(o, o')$  is the set of all transformation sequences that convert  $o$  into  $o'$ .
- *The dissimilarity* between  $o$  and  $o'$ , denoted  $\mathbf{dis}(o, o')$  w.r.t. a set  $TR$  of transformation operators, and a set  $CF$  of cost functions is given by:

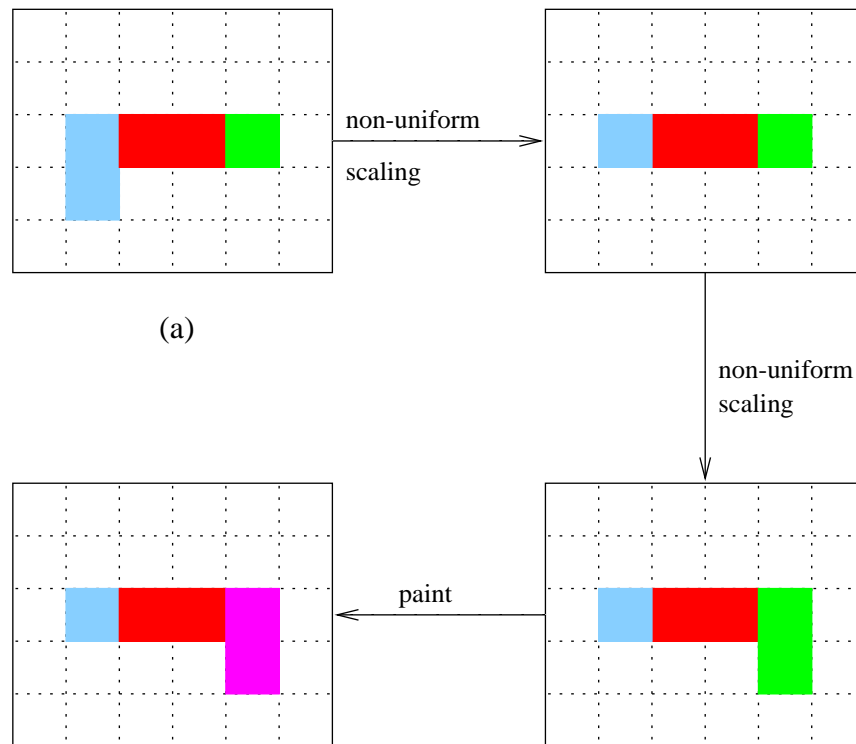
$$\mathbf{dis}(o, o') = \min\{\mathbf{cost}(TS) \mid TS \in \mathbf{TSeq}(o, o') \cup \mathbf{TSeq}(o', o)\}.$$

## Example



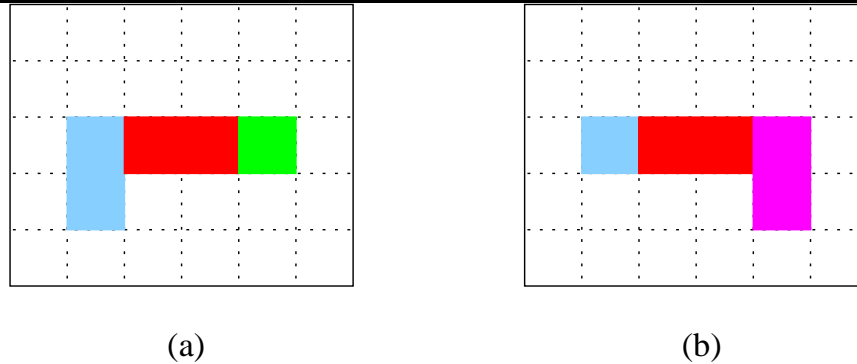
$TS_1$ : This transformation sequence consists of a

- non-uniform scaling operation (scale the blue part of  $o_1$  50% in the vertical upward direction, leaving the horizontal unchanged),
- a non-uniform scaling operation (scale the green part of object  $o_1$  by a 100% increase in the vertical, downward direction, with no change in the horizontal).
- The third operation applies the **paint** operation, painting the two pixels colored magenta to green.
- The Figure below depicts the intermediate steps.



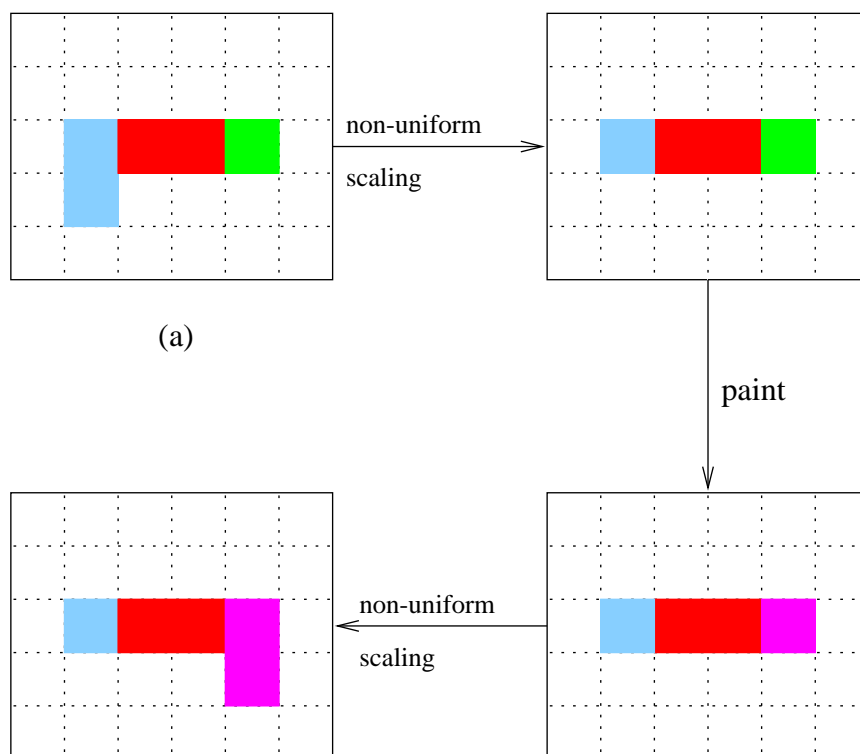


## Example



$TS_2$ : This transformation sequence consists of a

- non-uniform scaling operation (scale the blue part of  $o_1$  50% in the vertical upward direction, leaving the horizontal unchanged).
- apply the **paint** operation, painting the green object magenta.
- apply the non-uniform scaling operation (scale the magenta part of object  $o_1$  by a 100% increase in the vertical, downward direction, with no change in the horizontal).
- The following figure depicts the intermediate steps.



- If we assume that the cost functions associated with non-uniform scaling are independent of color, and the **paint** operation merely counts the number of pixels being painted, then it is easy to see that transformation  $TS_2$  accomplishes the desired transformation at a cheaper cost (as it paints one less pixel than transformation  $TS_1$ ).

## Transformation Model vs. the Metric Model

---

### Advantages of the Transformation Model over the Metric Model

- First and foremost, the user can “set up” his own notion of similarity by specifying that certain transformation operators may/may not be used.
- Second, the user may associate, with each transformation operator, a *cost function* that assesses a cost to each application of the operation, depending upon the arguments to the transformation operator. This allows the user to personalize the notion of similarity for his/her needs.

### Advantages of the Metric Model over the Transformation Model

By forcing the user to use one and only one (dis)similarity metric, the system can facilitate the indexing of data so as to optimize the one operation of finding the “nearest” neighbor (i.e. least dis-similar) object w.r.t. the query object specified by the user.

## Alternative Image DB Paradigms

---

- **Representing IDBs as Relations**
- **Representing IDBs with Spatial Data Structures**
- **Representing IDBs using Image Transformations**

Two different photographs of the same person may vary, depending upon a variety of factors such as:

1. the time of the day at which the two photographs were taken;
2. the lighting conditions under which the photographs were taken;
3. the camera used;
4. the exact position of the subject's head and his/her facial expression.
5. etc.

## Representing Image DBs with Relations

---

Suppose  $IDB = (GI, Prop, Rec)$ .

1. Create a relation called **images** having the scheme:

$(Image, ObjID, XLB, XUB, YLB, YUB)$

where **Image** is the name of an image file, **ObjID** is a dummy name created for an object contained in the image, and **XLB**, **XUB**, **YLB**, **YUB** describe the rectangle in question. If  $R$  is a rectangle specified by **XLB**, **XUB**, **YLB**, **YUB** and  $R$  is in  $Rec(I)$ , then there exists a tuple

$(I, newid, XLB, XUB, YLB, YUB)$

in the relation **images**.

2. For each property  $p \in Prop$ , create a relation  $R_p$  having the scheme:

$(Image, XLB, XUB, YLB, YUB, Value)$ .

Here, **Image** is the name of an image file. Unlike the preceding case though, **XLB**, **XUB**, **YLB**, **YUB** denote a rectangular *cell* in the image, and **Value** specifies the value of the property  $p$ .

### Example:

Image	Obj Id	XLB	XUB	YLB	YUB
pic1.gif	$o_1$	10	60	5	50
pic1.gif	$o_2$	80	120	20	55
pic2.gif	$o_3$	20	65	20	75
pic3.gif	$o_4$	25	75	10	60
pic4.gif	$o_5$	20	60	30	80
pic5.gif	$o_6$	0	40	15	50
pic6.gif	$o_7$	20	75	15	80
pic6.gif	$o_8$	20	70	130	185
pic7.gif	$o_9$	15	70	15	75

## Image Properties

---

- Pixel Level Properties: e.g. RGB values
- Object/Region Level Properties: e.g. NAME, AGE
- Image Level Properties: e.g. when image was captured, where, and by whom

## Querying (Relational Representations of) Image DBs

---

- Eliciting the contents of an image is done using image processing algorithms.
- Image processing algorithms are usually only partially accurate.
- This implies that tuples placed in a relation by an image processing program has certain associated probabilistic attributes.
- Each object has a value associated with each property  $p \in \text{Prop}$ .
- Each relation  $R_p$  associated with object/region level properties as well as image level properties has just two attributes: an object (or region) id, and a value for the property.
- Example:

Relation **name**



ObjId	Name
$o_1$	Jim Hatch
$o_2$	John Lee
$o_3$	John Lee
$o_4$	Jim Hatch
$o_5$	Bill Bosco
$o_6$	Dave Dashell
$o_7$	Ken Yip
$o_8$	Bill Bosco
$o_7$	Ken Yip

- This description must be enhanced to include probabilistic object identification.

## Probabilistic Version of the name relation

---

Probabilistic Version of the **name** relation

ObjId	Name	Prob
$o_1$	Jim Hatch	0.8
$o_1$	Dave Fox	0.2
$o_2$	John Lee	0.75
$o_2$	Ken Yip	0.15
$o_3$	John Lee	1
$o_4$	Jim Hatch	1
$o_5$	Bill Bosco	1
$o_6$	Dave Dashell	1
$o_7$	Ken Yip	0.7
$o_7$	John Lee	0.3
$o_8$	Bill Bosco	0.6
$o_8$	Dave Dashell	0.2
$o_8$	Jim Hatch	0.10
$o_9$	Ken Yip	1

Reading:

1. The probability that “John Lee” is the name attribute of  $o_2$  is 0.75.
2. The probability that “Ken Yip” is the name attribute of  $o_2$  is 0.15.
3. There is, in this case, a 10% missing probability.

## Complex Queries

---

- Suppose we ask the query *What is the probability that `pic1.gif` contains both Jim Hatch and Ken Yip?* Is the answer the product of the two probabilities, i.e. is it  $(0.8 \times 0.15) = 0.12$  ?
- *What is the probability that `pic6.gif` contains both Jim Hatch and Ken Yip?* Is the answer  $(0.7 \times 0.1) = 0.07$  ?
- In general, the answer is NO.

**Example:** Consider a hypothetical image `pic8.gif` with two objects  $o_{10}, o_{11}$  in it, and suppose our table above is expanded by the insertion of the following new tuples identified by the image processing algorithm.

ObjId	Name	Prob
$o_{10}$	Ken Yip	0.5
$o_{10}$	Jim Hatch	0.4.
$o_{11}$	Jim Hatch	0.8
$o_{11}$	John Lee	0.1

If we are ignorant about the dependencies between different events (as we are in the above case, then we are forced to confront *four possibilities*:

**Possibility 1**  $o_{10}$  is Ken Yip and  $o_{11}$  is John Hatch.

**Possibility 2**  $o_{10}$  is Ken Yip and  $o_{11}$  is *not* John Hatch.

**Possibility 3**  $o_{10}$  is *not* Ken Yip but  $o_{11}$  is John Hatch.

**Possibility 4**  $o_{10}$  is *not* Ken Yip and  $o_{11}$  is *not* John Hatch.

## Complex Queris

---

- Suppose  $p_i$  denotes the probability of Possibility  $i$ ,  $1 \leq i \leq 4$ . Then, we can say that:

$$p_1 + p_2 = 0.5.$$

$$p_3 + p_4 = 0.5.$$

$$p_1 + p_3 = 0.8.$$

$$p_2 + p_4 = 0.2$$

$$p_1 + p_2 + p_3 + p_4 = 1.$$

- The first equation follows from the fact that  $o_{10}$  is Ken Yip according to possibilities 1 and 2, and we know from the table, that the probability of  $o_{10}$  being Ken Yip is 0.5.
- The second equation follows from the fact that  $o_{10}$  is someone other than Ken Yip according to possibilities 3 and 4, and we know from the table, that the probability of  $o_{10}$  not being Ken Yip is 0.5.
- The third equation follows from the fact that  $o_{11}$  is Jim Hatch according to possibilities 1 and 3, and we know from the table, that the probability of  $o_{11}$  being Jim Hatch is 0.8.
- Finally, the last equation follows from the fact that  $o_{11}$  is someone other than Jim Hatch according to possibilities 2 and 4, and we know from the table, that the probability of  $o_{11}$  not being John Hatch is 0.2.

- In order to determine the probability that `pic8.gif` contains both Ken Yip and John Hatch, we must attempt to solve the above system of linear equations for  $p_1$ , keeping in mind the fact that all scenarios possible are covered by our four possibilities. The result we obtain, using a linear programming engine, is that  $p_1$ 's probability is *not uniquely determinable*. It could be as low as 0.3 or as high as 0.5, or anywhere in between. In particular, note that merely multiplying the probability of 0.5 associated with Ken Yip being object  $o_{10}$  and the probability value 0.8 of m Hatch being object  $o_{11}$  leads to a probability of 0.4 which is certainly inside this interval, but does not accurately capture the four possibilities listed above.
- Requires the use of interval probabilities.

## Interval Probability Model

---

Interval Probabilistic Version of the **name** relation with  
**pic8.gif** included

ObjId	Name	Prob (Lower)	Prob (Upper)
$o_1$	Jim Hatch	0.77	0.83
$o_1$	Dave Fox	0.17	0.23
$o_2$	John Lee	0.72	0.78
$o_2$	Ken Yip	0.12	0.18
$o_3$	John Lee	0.97	1.00
$o_4$	Jim Hatch	0.97	1.00
$o_5$	Bill Bosco	0.97	1.00
$o_6$	Dave Dashell	0.97	1.00
$o_7$	Ken Yip	0.67	0.73
$o_7$	John Lee	0.27	0.33
$o_8$	Bill Bosco	0.57	0.63
$o_8$	Dave Dashell	0.17	0.23
$o_8$	Jim Hatch	0.07	0.13
$o_9$	Ken Yip	0.97	1.00
$o_{10}$	Ken Yip	0.47	0.53
$o_{10}$	Jim Hatch	0.37	0.43
$o_{11}$	Jim Hatch	0.77	0.83
$o_{10}$	Hohn Lee	0.07	0.13

- “Find an image that contains both Ken Yip and Jim Hatch.”

- Let us re-examine the image `pic8.gif` and see what the probability of this image containing both Ken Yip and Jim Hatch is.
- Constraints generated:

$$0.47 \leq p_1 + p_2 \leq 0.53.$$

$$0.47 \leq p_3 + p_4 \leq 0.53.$$

$$0.77 \leq p_1 + p_3 \leq 0.83.$$

$$0.17 \leq p_2 + p_4 \leq 0.23.$$

$$p_1 + p_2 + p_3 + p_4 = 1.$$

- Solving the above linear program for minimal and maximal values of the variable  $p_1$ , we obtain 0.24 and 0.53, respectively.
- Beauty is that when implementing this, we can avoid solving the linear program altogether.



## A General Approach

---

- A *probabilistic relation* over a scheme  $(A_1, \dots, A_n)$  is an ordinary relation over the scheme  $(A_1, \dots, A_n, LB, UB)$  where the domain of the  $LB$  and  $UB$  attributes is the unit interval  $[0, 1]$  of real numbers.
- In particular, the relation **name** is a probabilistic relation that has three attributes:

$(\text{ImageId}, \text{ObjectId}, \text{Name})$

of the sort we have already seen thus far.

- The **name** relation satisfies some integrity constraints:

$$(\forall \mathbf{t}_1, \mathbf{t}_2) \mathbf{t}_1.\text{ObjId} = \mathbf{t}_2.\text{ObjId} \rightarrow \mathbf{t}_1.\text{ImageId} = \mathbf{t}_2.\text{ImageId}.$$

This constraint states that an `ObjectId` can be associated with only one image, i.e. distinct images have distinct `ObjectIds`. The following constraint says that the `LB` field of any tuple is always smaller than the `UB` field.

$$(\forall \mathbf{t}) \mathbf{t}.\text{LB} \leq \mathbf{t}.\text{UB}.$$

- An *image database* consists of a probabilistic relation called **name** of the above form, together with a set of *ordinary* (i.e. non-probabilistic) relations  $R_1, \dots, R_k$  corresponding to image properties.

## Membership Queries

---

- A *membership query* in an image database is a query of the form: *Find all images in the image database that contain objects named  $s_1, \dots, s_n$ .*
- **SELECT**                **ImageId**  
**FROM**                **name**  $T_1, \dots, T_n$   
**WHERE**                 $T_1.\text{Name} = s_1$  AND  $\dots$  AND  $T_n.\text{Name} = s_n$  AND  
                               $T_1.\text{ImageId} = T_2.\text{ImageId}$  AND  $\dots$  AND  
                               $T_1.\text{ImageId} = T_n.\text{ImageId}$ .

The *result of this membership query* is a table containing *three* fields – the **ImageId** fields that is explicitly listed in the query, a **LB** field, and a **UB** field.  $(im, \ell, u)$  is in the result iff for each  $1 \leq j \leq n$ , there exists a tuple  $t_j \in \text{name}$  such that:

1.  $t.\text{ImageId} = im$  and
2.  $t.\text{LB} = \ell_i$  and  $t.\text{UB} = u_i$  and
3.  $[\ell, u] = [\ell_1, u_1] \otimes [\ell_2, u_2] \otimes \dots \otimes [\ell_n, u_n]$

where

$$[x, y] \otimes [x', y'] = [\max(0, x + x' - 1), \min(y, y')].$$

## Other Queries

---

- ‘Find all people who have had deposits of over 9000 dollars, and who have been photographed with Denis Jones.
- |        |   |
|--------|---|
| SELECT | I.ImageId   |
| FROM   | name I, bank B  |
| WHERE  | I CONTAINS <i>B.name</i> , Denis Jones AND<br>B.trans=deposit AND B.amount > 9000 AND<br>B.name = I.name. |

## Representing Image DBs with R-Trees

---

1. Create a relation called **occursin** with two attributes

(**ImageId**, **ObjId**)

specifying which objects appear in which images.

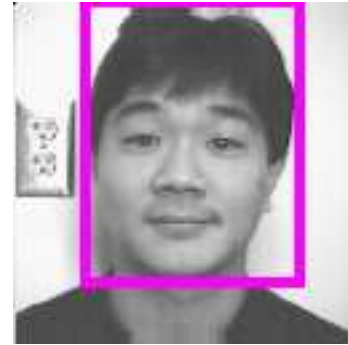
2. Create *one* R-tree that stores all the rectangles. If the same rectangle (say with  $XLB = 5$ ,  $XUB = 15$ ,  $YLB = 20$ ,  $YUB = 30$ ) appears in two images, then we have an overflow list associated with that node in the R-tree.
3. Each rectangle has an associated set of fields that specifies the object/region level properties of that rectangle.

## Example

---



pic1.gif



pic2.gif



pic3.gif



pic4.gif



pic5.gif



pic6.gif



pic7.gif

—V.S. Subrahmanian —All Rights Reserved—

These slides may not be duplicated without explicit written permission from Morgan Kaufmann Press.

Principles of Multimedia Database Systems   Morgan Kaufmann   Copyright ©1997

## occursin Relation

---

pic1.gif	$o_1$
pic1.gif	$o_2$
pic2.gif	$o_3$
pic3.gif	$o_4$
pic4.gif	$o_5$
pic5.gif	$o_6$
pic6.gif	$o_7$
pic6.gif	$o_8$
pic7.gif	$o_9$

## R-tree representation

---

```
facenode = record  
     $Rec_1, Rec_2, Rec_3$ : rectangle;  
     $P_1, P_2, P_3$ :  $\uparrow$ rtnodetype  
end
```

```
rectangle = record  
    XLB,XUB,YLB,YUB: integer;  
    objlist:  $\uparrow$ objnode;  
    day,mth, yr: integer;  
    camera_type: string;  
    place: string  
end
```

```
objnode = record  
    objid: string;  
    imageid: string;  
    info: infotype  
end
```

```
infotype = record  
    objname: string;  
    Lp, Up: real: (* lower and upper probability bounds  
*)  
    Next:  $\uparrow$ objinfo  
end
```



## R-Tree Construction

---

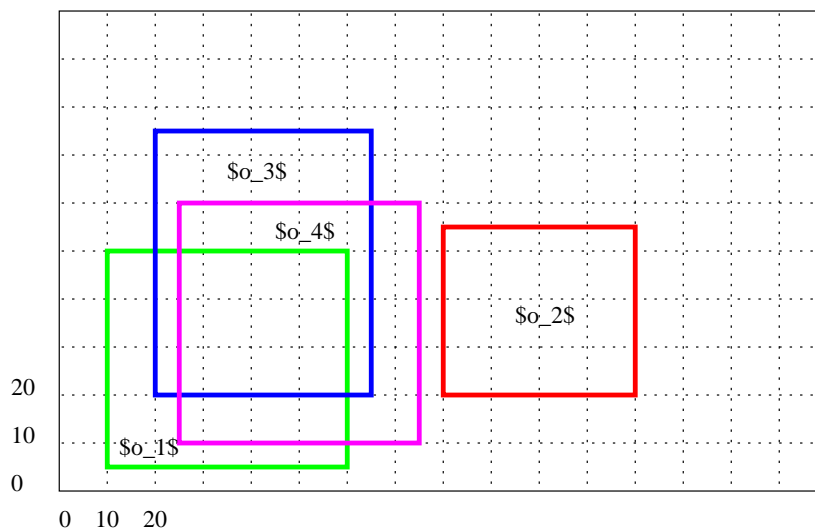
- Get Rectangles
- Create R-tree
- Flesh out Objects

## Get Rectangles

---

First and foremost, we may construct a small table describing all the rectangles that occur, and the images they occur in.

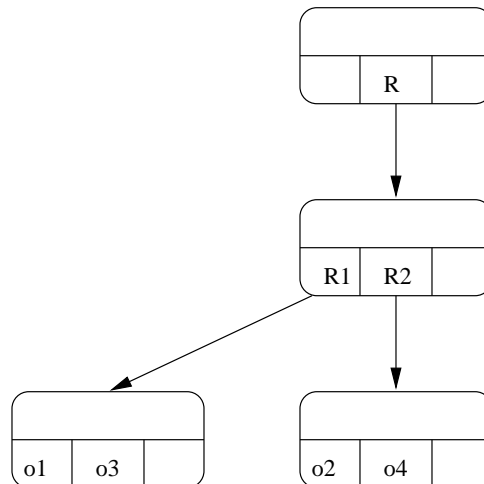
ObjId	ImageId	XLB	XUB	YLB	YUB
$o_1$	pic1.gif	10	60	5	50
$o_2$	pic1.gif	80	120	20	55
$o_3$	pic2.gif	20	65	20	75
$o_4$	pic3.gif	25	75	10	60



## Create R-Tree

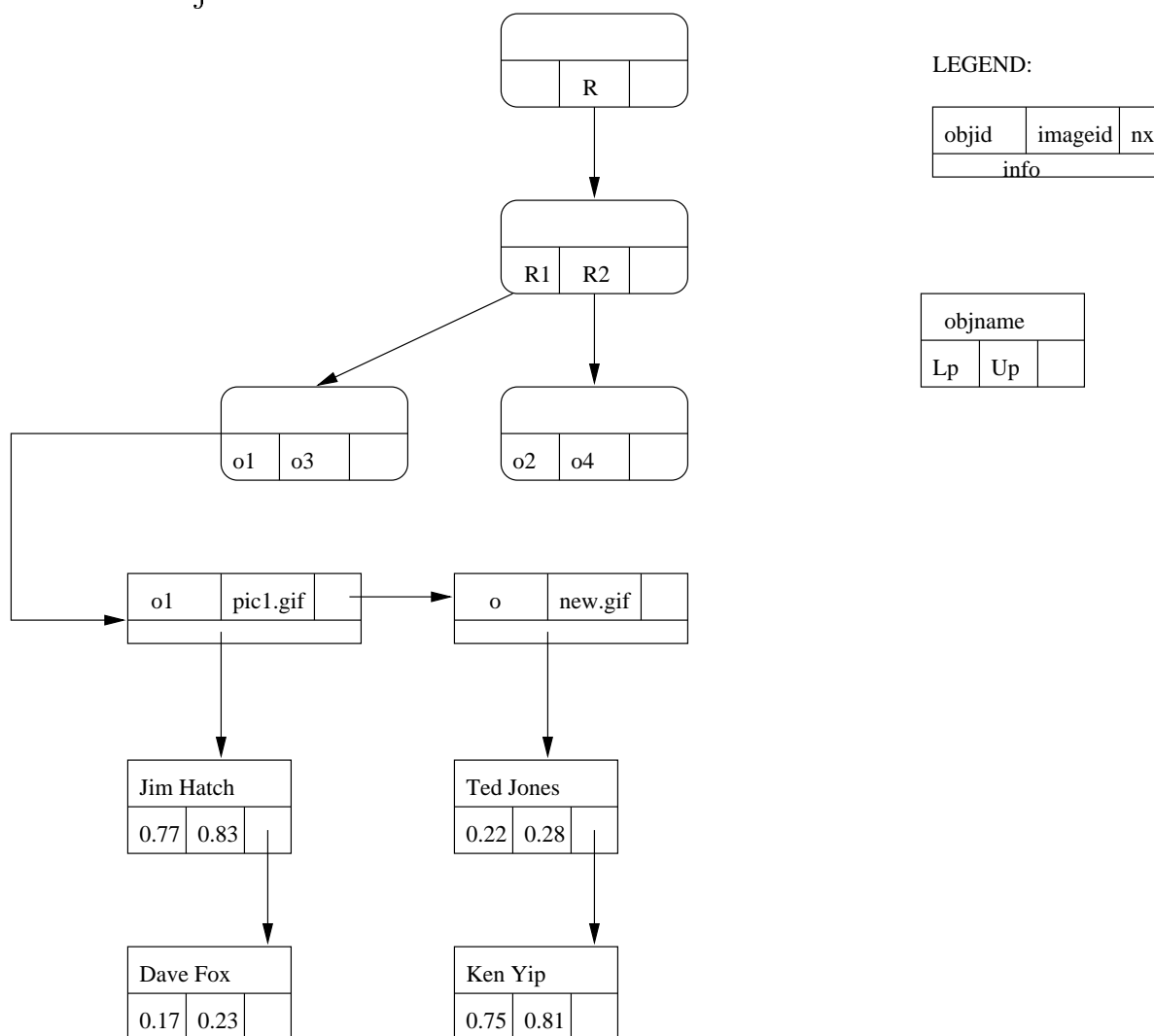
---

We then create an R-tree representing the above rectangles. At this stage the object nodes in the R-tree may still not be “filled in” completely.



## Flesh Out Objects

We then “flesh out” and appropriately fill in the fields of the various objects stored in the R-tree.



## Generalized R-trees

---

- Previous representation does not provide an efficient way to perform nearest neighbor searches.
- Why? Only 2-attributes (bounding rectangles) are stored.
- In general, an object  $o$  has an associated  $(n + 2)$ -dimensional vector.
- If each point in the 2-d rectangle has  $n$  attributes, then we need to use a generalized rectangle.
- A *generalized rectangle* for a space of dimensionality  $g$  (specifically consider  $g = n + 2$ ) may be defined by a set of constraints of the form:

$$\ell_1 \leq x_1 \leq u_1$$

$$\ell_2 \leq x_2 \leq u_2$$

$$\dots$$

$$\ell_g \leq x_g \leq u_g$$

- When  $g = 2$ , we have  $n = 0$ , and in this case, an ordinary 2-dimensional rectangle is a special case of this definition.
- Sets of generalized rectangles are represented by generalized  $R$ -trees, or gR-trees.

## gR-Trees

---

A generalized R-tree (gR-tree) of order  $K$  is exactly like an R-tree except for the following factors:

- First, each node  $N$  represents a generalized bounding rectangle  $GBR(N)$  of dimensionality  $(n + 2)$ , which is represented by  $2 \times (n + 2)$  real number fields, one for the lower bound and upper bound, respectively, of each dimension.
- When a node  $N$  is split, the union of the generalized bounding rectangles associated with its children equals the generalized bounding rectangle associated with  $N$ .
- Each node (other than the root and the leaves) contains at most  $K$  generalized bounding rectangles and at least  $\lceil K/2 \rceil$  generalized rectangles.
- As usual, all  $(n + 2)$ -dimensional “data” rectangles are stored in leaves.

## Nearest Rectangular Neighbors

---

- Suppose  $R_Q$  is a query rectangle (which may represent an image object).
- Find all rectangles in a gR-tree  $T$  that are as close to  $R_Q$  as possible (where closeness is defined by a metric  $d$  on points).
- We extend the metric  $d$  to apply to rectangles as follows:

$$d(R, R') = \min\{d(p, p') \mid p \in R, p' \in R'\}.$$

# Algorithm

---

**Algorithm 1** *NN\_Search\_GR*( $T, R_Q$ )

```

SOL = NIL; ( $\star$  no solution so far  $\star$ );
Todo = List containing T only;
Bestdist =  $\infty$ ; ( $\star$  distance of best solution from  $R_Q$   $\star$ );
while Todo  $\neq$  NIL do
{
    F = first element of Todo;
    Todo = delete F from Todo;
    if  $d(GBR(F), R_Q) < Bestdist$  then
    {
        Compute children  $N_1, \dots, N_r$  of F;
        if  $N_i$ 's are leaves of T then
        {
             $N_{min}$  = any  $N_i$  at minimal distance from  $R_Q$ ;
             $Ndist = d(GBR(N_i), R_Q)$ ;
            if  $Ndist < Bestdist$  then
            {
                 $Bestdist = Ndist$ ;  $SOL = N_{min}$ ;
            }
        }
    }
    else Todo = insert all  $N_i$ 's into Todo in order of distance from  $R_Q$ ;
}
Return SOL;
end

```





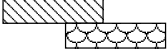
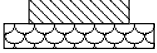
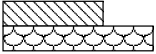
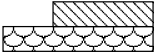
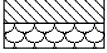
## Retrieving Images by Spatial Layout

---

Given an image  $I$ , and two objects (represented by rectangles)  $o1$ ,  $o2$  in  $I$ , a user may wish to ask queries of the form:

1. Is  $o1$  to the South of  $o2$  ?
2. Is  $o1$  to the South-East of  $o2$  ? item Is  $o1$  to the left of  $o2$  ?
3. Are  $o1$  and  $o2$  overlapping ?

## One-dimensional Precedence Relations

o1	o2	Description	Notation
		o1 before o2	B(o1,o2)
		o1 meets o2	M(o1,o2)
		o1 overlaps o2	OV(o1,o2)
		o1 during o2	D(o1,o2)
		o1 starts o2	S(o1,o2)
		o1 finishes o2	F(o1,o2)
		o1 equal o2	EQ(O1,o2)

## 2-Dimensional Precedence Relations

---

It is easy to extend the precedence relation along one dimension, to the case of two dimensions, is straightforward. If we use the notation  $o[x]$  and  $o[y]$  to denote the projection of object  $o$  on the  $x$  and  $y$  axes, respectively, then it is easy to capture our spatial relationships as follows:

1. We say  $o1$  is **South** of  $o2$  iff  $B(o1[y], o2[y])$  and either (i)  $D(o1[x], o2[x])$  or (ii)  $D(o2[x], o1[x])$  or (iii)  $S(o1[x], o2[x])$  or (iv)  $S(o2[x], o1[x])$  holds or (v)  $F(o1[x], o2[x])$  or (vi)  $F(o2[x], o1[x])$  holds or (vii)  $E(o1[x], o2[x])$  holds.
2. Likewise, we say that  $o1$  is to the **Left** of  $o2$  iff either (i)  $B(o1[x], o2[x])$  holds or (ii)  $M(o1[x], o2[x])$  holds.

# Text/Document Databases

---

## What is a text database?

---

- User wants to find documents related to a topic  $T$ .
- The search program tries to find the documents in the “document database” that contain the string  $T$ .
- This has two problems:
  1. **Synonymy:** Given a word  $T$  (i.e. specifying a topic), the word  $T$  does not occur anywhere in a document  $D$ , even though the document  $D$  is in fact closely related to the topic  $T$  in question.
  2. **Polysemy:** The same word may mean many different things in different contexts.
- Consider a text database that only indexes the following titles.

DocumentID	String
$d_1$	Jose Orojuelo's Operations in Bosnia.
$d_2$	The Medellin Cartel's Financial Organization.
$d_3$	The Cali Cartel's Distribution Network.
$d_4$	Banking Operations and Money Laundering.
$d_5$	Profile of Hector Gomez.
$d_6$	Connections between Terrorism and Asian Dope Operations.
$d_7$	Hector Gomez's: How he Gave Agents the Slip in Cali.
$d_8$	Sex, Drugs, and Videotape.
$d_9$	The Iranian Connection.
$d_{10}$	Boating and Drugs: Slips owned by the Cali Cartel.

## Organization of this Topic

---

- Measures of performance of a text retrieval system.
- Latent Semantic Indexing
- Telescopic-Vector Trees for Document Retrieval.

## Precision and Recall

---

- Suppose  $D$  is a finite set of documents.
- Suppose  $\mathcal{A}$  is any algorithm that takes as input, a topic string  $T$ , returns as output, a set ( $\mathcal{T}$ ) of documents.
- **Precision** of algorithm  $\mathcal{A}$  w.r.t. the predicate *relevant* and test set  $D_{test}$  is  $P_t\%$  for topic  $t \in T_{test}$  iff:

$$P_t = 100 \times \frac{1 + \text{card}(\{d \in D_{test} \mid d \in \mathcal{A}(t) \wedge \text{relevant}(t, d) \text{ is true}\})}{1 + \text{card}(\{d \in D_{test} \mid d \in \mathcal{A}(t)\})}.$$

(To avoid division by zero, we add one to both the numerator and denominator above). We say that the precision of algorithm  $\mathcal{A}$  w.r.t. the predicate *relevant*, the document test set  $D_{test}$  and the topic test set  $T_{test}$  is  $P\%$  iff:

$$P = \frac{\sum_{t \in T_{test}} P_t}{\text{card}(T_{test})}.$$

- Precision basically says: How many of the answers returned are in fact correct.
- **Recall** of an algorithm  $\mathcal{A}$  is a measure of “how many” of the right documents are in fact retrieved by the query.
- Rrecall  $R_t$  associated with a topic  $t$  is given by the formula:

$$R_t = 100 \times \frac{1 + \text{card}(\{d \in D_{test} \mid d \in \mathcal{A}(t) \wedge \text{relevant}(t, d) \text{ is true}\})}{1 + \text{card}(\{d \in D_{test} \mid \text{relevant}(t, d) \text{ is true}\})}.$$

The overall recall rate  $R$  associated with test sets  $D_{test}$  of documents, and  $T_{test}$  of topics, is given by:

$$R = \frac{\sum_{t \in T_{test}} R_t}{card(T_{test})}.$$



## Stop Lists, Word Stems, and Frequency Tables

---

- **Stop List:** This is a set of words that do not “discriminate” between the documents in a given archive.
- E.g. Cornell SMART system has about 440 words on its stop list.
- **Word Stems:** Many words are small syntactic variants of each other. For example, the words **drug**, **drugged**, **drugs**, are all similar in the sense that they share a common “stem”, viz. the word **drug**.
- Most document retrieval systems first eliminate words on stop lists and they also reduce words to their stems, before creating a frequency table.
- **Frequency Tables:** Suppose
  - $D$  is a set of  $N$  documents, and
  - $T$  is a set of  $M$  words/terms occurring in the documents of  $D$ .
  - Assume that no words on the stop list for  $D$  occur in  $T$ , and that all words in  $T$  have been stemmed.

The *frequency table*, **FreqT**, associated with  $D$  and  $T$  is an  $(M \times N)$  matrix such that **FreqT**( $i, j$ ) equals the number of occurrences of the word  $t_i$  in document  $d_j$ .

## Example

---

DocumentID	String
$d_1$	Jose Orojuelo's Operations in Bosnia.
$d_2$	The Medellin Cartel's Financial Organization.
$d_3$	The Cali Cartel's Distribution Network.
$d_4$	Banking Operations and Money Laundering.
$d_5$	Profile of Hector Gomez.
$d_6$	Connections between Terrorism and Asian Dope Operations.
$d_7$	Hector Gomez's: How he Gave Agents the Slip in Cali.
$d_8$	Sex, Drugs, and Videotape.
$d_9$	The Iranian Connection.
$d_{10}$	Boating and Drugs: Slips owned by the Cali Cartel.

The associated frequency table might be given by:

Term/Doc	$d_8$	$d_9$	$d_{10}$	$d_{11}$
sex	1	0	0	0
drug	1	0	1	3
videotape	1	0	0	0
iran	0	1	0	0
connection	0	1	0	0
boat	0	0	1	0
slip	0	0	1	0
own	0	0	1	0
cali	0	0	1	0
cartel	0	0	1	0

## Another example

---

Consider the frequency table shown below.

Term/Doc	$d_1$	$d_2$	$d_3$	$d_4$	$d_5$	$d_6$
$t_1$	615	390	10	10	18	65
$t_2$	15	4	76	217	91	816
$t_3$	2	8	815	142	765	1
$t_4$	312	511	677	11	711	2
$t_5$	45	33	516	64	491	59

- $d_1, d_2$  are similar because the distribution of the words in  $d_1$  “mirrors” the distribution of the words for  $d_2$ .
- Both contain lots of occurrences of  $t_1, t_4$  and relatively few occurrences of  $t_2, t_3$ , and moderately many occurrences of  $t_5$ .
- $d_3$  and  $d_5$  are also similar.
- But  $d_4, d_6$  stand out as sharply different.
- But is this enough?
- Merely counting words does not indicate the importance of the words, in the document. What about document lengths?
- Usually, a frequency table represents not just the number of occurrences of a (stemmed) word in a document, but also its importance.

## Queries

---

- User wants to execute the query: *Find the 25 documents that are maximally relevant w.r.t. banking operations and drugs?*.
- Query  $Q$  is trying to retrieve documents relevant to two keywords, which after “stemming” are:

**drug, bank.**

- Think of the query  $Q$  as a document. Thus,  $Q$  is a column vector.
- We want to find the columns in **FreqT** that are “as close” as possible to the vector associated with  $Q$ .
- Example closeness metrics include:

1. **Term Distance:** Suppose  $vec_Q(i)$  denotes the number of occurrences of term  $t_i$  in  $Q$ . Then the *term distance* between  $Q$  and document  $d_r$  is given by:

$$\sqrt{\sum_{j=1}^M (vec_Q(j) - \text{FreqT}(j, r))^2}.$$

Of course, this is a rather arbitrary metric.

2. **Cosine Distance:** This metric is used extensively in the document database world, and it may be described as follows:

$$\frac{\sum_{j=1}^M (vec_Q(j) \times \text{FreqT}(j, r))}{\sqrt{\sum_{j=1}^M vec_Q(j)^2} \times \sqrt{\sum_{j=1}^M \text{FreqT}(j, r)^2}}.$$

- Complexity of retrievals may be  $O(N \times M)$  which could be staggering in size.

## Latent Semantic Indexing: The Basic Idea

---

- The number of documents  $M$  and the number of terms  $N$  is very large. For example,  $N$  could be over 10,000,000, as English words as well as proper nouns can be indexed.
- What LSI tries to do is to find a relatively small subset of  $K$  words which discriminate between the  $M$  documents in the archive.
- LSI is claimed to work effectively for around  $K = 200$ .
- Advantage: Each document is now a column vector of length 200, instead of length  $N$ .
- This is obviously a big plus.
- Bottom line: *How do we find a relatively small subset of  $K$  words which discriminate between the  $M$  documents in the archive.*
- Use a technique called singular valued decomposition.
- 4-step approach used by LSI:
  1. **(Table Creation)** Create frequency matrix **FreqT**.
  2. **(SVD Construction)** Compute the singular valued decompositions,  $(A, S, B)$  of **FreqT**.
  3. **(Vector Identification)** For each document  $d$ , let  $vec(d)$  be the set of all terms in **FreqT** whose corresponding rows have not been eliminated in the singular matrix  $S$ .

4. **(Index Creation)** Store the set of all  $vec(d)$ 's, indexed by any one of a number of techniques (later we will discuss one such technique called a TV-tree).

## Singular Valued Decompositions

---

- A matrix  $\mathcal{M}$  is said to be of order  $(m \times n)$  if it has  $m$  rows and  $n$  columns.
- If  $\mathcal{M}_1, \mathcal{M}_2$  are matrices of order  $(m_1 \times n_1)$  and  $(m_2 \times n_2)$  respectively, then we say that the product,  $(\mathcal{M}_1 \times \mathcal{M}_2)$ , is *well defined* iff  $n_1 = m_2$ .
- If:

$$\mathcal{M}_1 = \begin{pmatrix} a_1^1 & a_2^1 & \cdots & a_{m_1}^1 \\ a_1^2 & a_2^2 & \cdots & a_{m_1}^2 \\ \cdots & \cdots & \cdots & \cdots \\ a_1^{n_1} & a_2^{n_1} & \cdots & a_{m_1}^{n_1} \end{pmatrix}; \mathcal{M}_2 = \begin{pmatrix} b_1^1 & b_2^1 & \cdots & b_{m_2}^1 \\ b_1^2 & b_2^2 & \cdots & b_{m_2}^2 \\ \cdots & \cdots & \cdots & \cdots \\ b_1^{n_2} & b_2^{n_2} & \cdots & b_{m_2}^{n_2} \end{pmatrix}$$

then the product,  $(\mathcal{M}_1 \times \mathcal{M}_2)$  is the matrix

$$(\mathcal{M}_1 \times \mathcal{M}_2) = \begin{pmatrix} c_1^1 & c_2^1 & \cdots & c_{m_1}^1 \\ c_1^2 & c_2^2 & \cdots & c_{m_1}^2 \\ \cdots & \cdots & \cdots & \cdots \\ c_1^{n_2} & c_2^{n_2} & \cdots & c_{m_1}^{n_2} \end{pmatrix}$$

where:

$$c_j^i = \sum_{r=1}^{n_1} (a_r^i \times b_j^r).$$

- EX:

$$\begin{pmatrix} 3 & 2 \\ 4 & 8 \end{pmatrix} \times \begin{pmatrix} 1 & 4 & 3 \\ 2 & 4 & 6 \end{pmatrix} = \begin{pmatrix} 7 & 20 & 21 \\ 20 & 48 & 60 \end{pmatrix}.$$



## SVD Continued

---

- Given a matrix  $\mathcal{M}$  of order  $(m \times n)$ , the *transpose* of  $\mathcal{M}$ , denoted  $\mathcal{M}^T$ , is obtained by converting each row of  $\mathcal{M}$ , into a column of  $\mathcal{M}^T$ .

- EX:

$$\begin{pmatrix} 7 & 20 & 21 \\ 20 & 48 & 60 \end{pmatrix}^T = \begin{pmatrix} 7 & 20 \\ 20 & 48 \\ 21 & 60 \end{pmatrix}.$$

- Vector = matrix of order  $(1 \times m)$ .
- Two vectors  $x, y$  of the same order are said to be *orthogonal* iff  $x^T y = 0$ .
- EX:

$$x = (10, 5, 20).$$

$$y = (1, 2, -1).$$

These two vectors are orthogonal because

$$x^T y = \begin{pmatrix} 10 \\ 5 \\ 20 \end{pmatrix} \times (1 \quad 2 \quad -1) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

## SVD Continued

---

- Matrix  $\mathcal{M}$  is said to be **orthogonal** iff  $(\mathcal{M}^T \times \mathcal{M})$  is the identity matrix (i.e. the matrix, all of whose entries are 1). For example, consider the matrix:

$$\mathcal{M} = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}.$$

This matrix is othogonal.

- Matrix  $\mathcal{M}$  is said to be a *diagonal matrix* iff the order of  $\mathcal{M}$  is  $(m \times m)$  and for all  $1 \leq i, j \leq m$ , it is the case that:

$$i \neq j \rightarrow \mathcal{M}(i, j) = 0.$$

- EX:  $A$  and  $B$  below are diagonal matrices, but  $C$  is not:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 5 \end{pmatrix}; B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}; C = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}.$$

- Diagonal matrix  $\mathcal{M}$  of order  $(m \times m)$  is said to be *non-decreasing* iff for all  $1 \leq i, j \leq m$ ,

$$i \leq j \rightarrow \mathcal{M}(i, i) \leq \mathcal{M}(j, j).$$

- Above,  $A$  is a non-decreasing diagonal matrix, but  $B$  is not.

## SVD Continued

---

- A *Singular Value Decomposition* of  $\text{FreqT}$  is a triple  $(A, S, B)$  where:
  1.  $\text{FreqT} = (A \times S \times B^T)$  and
  2.  $A$  is an  $(M \times M)$  orthogonal matrix such that  $A^T A = I$  and
  3.  $B$  is an  $(N \times N)$  orthogonal matrix such that  $B^T B = I$  and
  4.  $S$  is a diagonal matrix called a *singular matrix*.
- Theorem: Given any matrix  $\mathcal{M}$  of order  $(m \times n)$ , it is possible to find a singular value decomposition,  $(A, S, B)$  of  $\mathcal{M}$  such that  $S$  is a *non-decreasing* diagonal matrix.
- EX: The SVD of the matrix

$$\begin{pmatrix} 1.44 & 0.52 \\ 0.92 & 1.44 \end{pmatrix}$$

is given by:

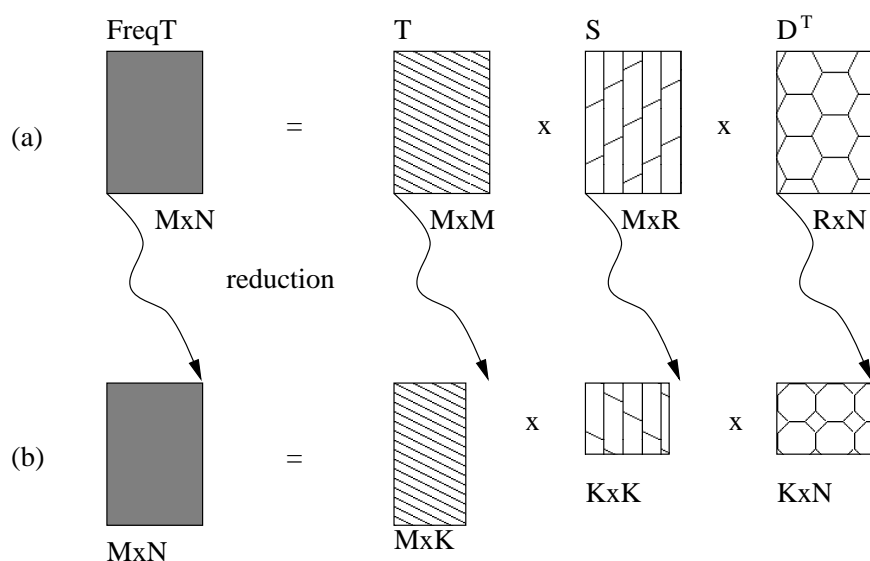
$$\begin{pmatrix} 0.6 & -0.8 \\ 0.8 & 0.6 \end{pmatrix} \begin{pmatrix} 5 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 0.8 & 0.6 \\ 0.6 & -0.8 \end{pmatrix}.$$

Here, the singular values are 5 and 2, and it is easy to see that the singular matrix is non-decreasing.

## Returning to LSI

---

- Given a frequency matrix  $\mathbf{FreqT}$ , we can decompose it into an SVD  $\mathcal{T}\mathcal{S}\mathcal{D}^T$  where  $\mathcal{S}$  is non-decreasing.
- If  $\mathbf{FreqT}$  is of size  $(M \times N)$ , then  $\mathcal{T}$  is of size  $(M \times M)$  and  $\mathcal{S}$  is of order  $(M \times R)$  where  $R$  is the rank of  $\mathbf{FreqT}$ , and  $\mathcal{D}^T$  is of order  $(R \times N)$ .
- We can now “shrink” the problem substantially by eliminating the least significant singular values from the singular matrix  $\mathcal{S}$ .



## LSI Continued

---

Shrinking the matrices is done as follows.

- Choose an integer  $k$  that is substantially smaller than  $R$ .
- Replace  $\mathcal{S}$  by  $\mathcal{S}^*$ , which is a  $(k \times k)$  matrix, such that  $\mathcal{S}^*(i, j) = \mathcal{S}(i, j)$  for  $1 \leq i, j \leq k$ .
- Replace the  $(R \times N)$  matrix  $\mathcal{D}^T$  by the  $(k \times N)$  matrix  $\mathcal{D}^{*T}$  where:  $\mathcal{D}^{*T}(i, j) = \mathcal{D}^T(i, j)$  if  $1 \leq i \leq k$  and  $1 \leq j \leq N$ .

Bottom line:

- Throw away the least significant values, and retain the rest of the matrix involved.
- Key claim in LSI Is that if  $k$  is chosen judiciously, then the  $k$  rows appearing in the singular matrix  $\mathcal{S}^*$  represent the  $k$  “most important” (from the point of view of retrieval) terms occurring in the entire document collection.

## Example

---

Suppose **FreqT** has the SVD

$$\begin{pmatrix} a_1^1 & a_2^1 & a_3^1 & a_4^1 & a_5^1 \\ a_1^2 & a_2^2 & a_3^2 & a_4^2 & a_5^2 \\ \dots & \dots & \dots & \dots & \dots \\ a_1^M & a_2^M & a_3^M & a_4^M & a_5^M \end{pmatrix} \begin{pmatrix} 20 & 0 & 0 & 0 & 0 \\ 0 & 16 & 0 & 0 & 0 \\ 0 & 0 & 12 & 0 & 0 \\ 0 & 0 & 0 & 0.08 & 0 \\ 0 & 0 & 0 & 0 & 0.004 \end{pmatrix} \begin{pmatrix} b_1^1 & b_2^1 & b_3^1 & \dots & b_N^1 \\ b_1^2 & b_2^2 & b_3^2 & \dots & b_N^2 \\ \dots & \dots & \dots & \dots & \dots \\ b_1^5 & b_2^5 & b_3^5 & \dots & b_N^5 \end{pmatrix}$$

- If we set 3 as the threshold, then we obtain the following result (after eliminating the 4'th and 5'th singular values.

$$\begin{pmatrix} a_1^1 & a_2^1 & a_3^1 \\ a_1^2 & a_2^2 & a_3^2 \\ \dots & \dots & \dots \\ a_1^M & a_2^M & a_3^M \end{pmatrix} \begin{pmatrix} 20 & 0 & 0 \\ 0 & 16 & 0 \\ 0 & 0 & 12 \end{pmatrix} \begin{pmatrix} b_1^1 & b_2^1 & b_3^1 & \dots & b_N^1 \\ b_1^2 & b_2^2 & b_3^2 & \dots & b_N^2 \\ b_1^3 & b_2^3 & b_3^3 & \dots & b_N^3 \end{pmatrix}$$

## Analysis

---

- Usually,  $R$  is taken to be 200.
- The size of the original frequency table is  $(M \times N)$  where  $M$  is the number of terms, and  $N$  is the number of documents. We may easily have  $M = 1,000,000$  and  $N = 10,000$ , even for just a small document database such as that consisting of the University of Maryland's Computer Science technical reports.
- The size of the three matrices, *after* we have reduced the size of the singular matrix to, say 200, is:
  - The first matrix's size is  $M \times R$ . With the above numbers, this is  $1000000 \times 200 = 200,000,000$ .
  - The singular matrix's size is  $200 \times 200 = 400,000$ . (In fact, of these 400,000 entries, only 200 at most need to be stored, as all other entries are zero).
  - The last matrix's size is  $R \times N$ . With the above numbers, this is  $200 \times 10000$ .

Adding up the above, we get a total of 202,000,200 entries in the tables after SVD's are applied. This is approximately 200 million.

- In contrast,  $(M \times N)$  is close to 10,000-million: in other words, the SVD trick reduced the space utilized to about  $\frac{1}{50}$ 'th of that required by the original frequency table.

## LSI: Document Retrieval using SVDs

---

Two questions:

1. Given two documents  $d_1, d_2$  in the archive, how “similar” are they?
2. Given a query string/document  $Q$ , what are the  $n$  documents in the archive that are “most relevant” for the query ?

- Suppose  $\mathbf{x} = (x_1, \dots, x_w)$  and  $\mathbf{y} = (y_1, \dots, y_w)$ .
- *Dot product* of  $\mathbf{x}$  and  $\mathbf{y}$ , denoted  $\mathbf{x} \odot \mathbf{y}$  is given by

$$\mathbf{x} \odot \mathbf{y} = \sum_{i=1}^w x_i \times y_i.$$

- The similarity of these two documents w.r.t. the SVD representation,  $\mathcal{TS}^* \times \mathcal{D}^{*T}$ , of a frequency table is given by computing the dot product of the two *columns* in the matrix  $\mathcal{D}^{*T}$  associated with these two documents.

$$\sum_{z=1}^R \mathcal{D}^{*T}[i, z] \times \mathcal{D}^{*T}[j, z].$$

- When we are asked to find the top  $p$  matches for  $Q$ , we are trying to find  $p$  documents  $d_{\alpha(1)}, \dots, d_{\alpha(p)}$  such that:
  1. for all  $1 \leq i \leq j \leq p$ , the similarity between  $vec_Q$  and  $d_{\alpha(i)}$  is greater than or equal to the similarity between  $vec_Q$  and  $d_{\alpha(j)}$ .



2. There is no other document  $d_z$  such that the similarity between  $d_z$  and  $vec_Q$  exceeds that of  $d_{\alpha(p)}$ .
- This can be done by using any generalized, high dimensional data structure that supports nearest neighbor searches.

## Telescopic Vector (TV) Trees

---

- Access to point data in very large dimensional spaces should be highly efficient.
- A document  $d$  may be viewed as a vector  $\vec{d}$  of length  $k$ , where the singular valued matrix, after decomposition, is of size  $(k \times k)$ .
- Thus, each document may be thought of as a point in a  $k$ -dimensional space.
- A *document database* may be thought of as a collection of such points, indexed appropriately.
- When a user  $u$  presents a query  $Q$ , s/he is in effect specifying, a vector  $vec(Q)$  of length  $k$ . We must find the  $p$  documents in the database that are maximally relevant to  $Q$ .
- This boils down to attempting to find the  $k$ -nearest neighbors present in the document database, of the query  $Q$ .
- The TV-tree is a data structure that borrows from R-trees in this effort.
- The TV-tree attempts to *dynamically and flexibly* decide how to branch, based on the data that is being examined. If lots of vectors all agree on certain attributes (e.g. if lots of documents all have many common terms), then we must organize our index by branching on those terms (i.e. fields of the vectors) that *distinguish* between these vectors/documents.

## Organization of a TV-Tree

---

- **NumChild**: This is the maximal number of children that any node in the TV-tree is allowed to have.
- $\alpha$ :  $\alpha$  is a number, greater than 0 and less than  $k$ , called the *number of active dimensions*.
- $\text{TV}(k, \text{NumChild}, \alpha)$  denotes TV-tree used to store  $k$ -dimensional data with **NumChild** as the maximal number of children, and  $\alpha$  as the number of active dimensions.
- Each node in a TV-tree represents a region. For this purpose, each node  $N$  in a TV-tree contains three fields:
  1.  $N.Center$ : This represents a point in  $k$ -dimensional space.
  2.  $N.Radius$ : This is a real number greater than 0.
  3.  $N.ActiveDims$ : This is a list of at most  $\alpha$  dimensions. Each of these dimensions is a number between 1 and  $k$ . Thus,  $N.ActiveDims$  is a subset of  $\{1, \dots, k\}$  of cardinality  $\alpha$  or less,

## Region associated with a node $N$

---

- Suppose  $\mathbf{x}$  and  $\mathbf{y}$  are points in  $k$ -dimensional space, and *ActiveDims* is some set of active dimensions. The *active distance* between  $\mathbf{x}$  and  $\mathbf{y}$ , denoted  $\text{act\_dist}(\mathbf{x}, \mathbf{y})$  is given by:

$$\text{act\_dist}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i \in \text{ActiveDims}} \mathbf{x}_i^2 - \mathbf{y}_i^2}.$$

Here,  $\mathbf{x}_i$  and  $\mathbf{y}_i$  denote the value of the  $i$ 'th dimension of  $\mathbf{x}$  and  $\mathbf{y}$ , respectively.

- EX:  $k = 200$  and  $\alpha = 5$  and the set *ActiveDims* =  $\{1, 2, 3, 4, 5\}$ . Suppose:

$$\mathbf{x} = (10, 5, 11, 13, 7, x_6, x_7, \dots, x_{200}).$$

$$\mathbf{y} = (2, 4, 14, 8, 6, y_6, y_7, \dots, y_{200}).$$

Then the active distance between  $\mathbf{x}$  and  $\mathbf{y}$  is given by:

$$\begin{aligned} \text{act\_dist}(\mathbf{x}, \mathbf{y}) &= \sqrt{(10 - 2)^2 + (5 - 4)^2 + (11 - 14)^2 + (13 - 8)^2 + (7 - 6)^2} \\ &= \sqrt{100} \\ &= 10. \end{aligned}$$

- Node  $N$  represents the region containing all points  $\mathbf{x}$  such that the active distance (w.r.t. the active dimensions in  $N.\text{ActiveDims}$ ) between  $\mathbf{x}$  and  $N.\text{Center}$  is less than or equal to  $N.\text{Radius}$ .

- EX: If we had a node  $N$  with its center at

$$N.Center = (10, 5, 11, 13, 7, 0, 0, 0, 0, \dots, 0)$$

and  $N.ActiveDims = \{1, 2, 3, 4, 5\}$ , then this node represents the region consisting of all points  $\mathbf{x}$  such that:

$$\sqrt{(\mathbf{x}_1 - 10)^2 + (\mathbf{x}_2 - 5)^2 + (\mathbf{x}_3 - 11)^2 + (\mathbf{x}_4 - 13)^2 + (\mathbf{x}_5 - 7)^2} \leq N.Radius$$

We use the notation  $Region(N)$  to denote the region represented by a node  $N$  in a TV-tree.

- $N$  also contains an array, **Child** of **NumChild** pointers to other nodes of the same type.

## Properties of TV-Trees

---

- All data is stored at the leaf nodes;
- Each node in a TV-tree (except for the root and the leaves) must be at least half full, i.e. at least half the **Child** pointers must be non-NIL.
- If  $N$  is a node, and  $N_1, \dots, N_r$  are all its children, then

$$Region(N) = \bigcup_{i=1}^r Region(N_i).$$

## Insertion into TV-trees

---

There are three key steps.

1. **Branch Selection:** The first operation is called branch selection. When we insert a new vector into the TV-tree, and we are at node  $N$  (with children  $N_i$ , for  $1 \leq i \leq \text{NumChild}$ ), we need to determine which of these children to insert the key into.
2. **Splitting:** The second approach is what to do, when we are at a leaf node that is full and cannot accommodate the vector  $\mathbf{v}$  we are inserting. This causes a split in that node.
3. **Telescoping:** Suppose a node  $N$  is split into subnodes  $N_1, N_2$ . In this case, it may well turn out that the vectors in  $\text{Region}(N_1)$  all agree on not just the active dimensions of the parent  $N$ , but a few more as well. The addition of these extra dimensions is called *telescoping*. Telescoping may also involve the *removal* of some active dimensions, as we shall see later.

## Example

---

- 5-dimensional space.
- $TV(5, 3, 2)$ .
- Space is a sphere centered at  $(0, 0, 0, 0, 0)$  with radius 50.
- Initially, tree is empty.
- Insert  $(5, 3, 20, 1, 5)$ . This is handled straightforwardly by the creation of a root node with
  1.  $Root.Center = (0, 0, 0, 0, 0)$ ;
  2.  $Root.Radius = 50$ .
  3. In this case, the root is also a leaf, with a pointer to the information relevant to the point  $\mathbf{v}_1 = (5, 3, 20, 1, 5)$ .
  4. Suppose  $Root.ActiveDims = \{2, 3\}$ .

See (a).

- Insert  $\mathbf{v}_2 = (0, 0, 18, 42, 4)$ .
- Insert  $\mathbf{v}_3 = (0, 0, 19, 39, 6)$ . At this stage, the root is “full”.
- Insert  $\mathbf{v}_4 = (9, 10, 2, 0, 16)$ .
  1. We must *split* the root.
  2. Take the four vectors involved and “group” them together into two groups, say.  $v_1, v_4$  and  $v_2, v_3$ . See figure (d).
  3. Insert  $\mathbf{v}_5 = (18, 5, 27, 9, 9)$ . Branch selection needed to determine how to branch. See Figure 2(a).



4. Insert  $\mathbf{v}_6 = (0, 0, 29, 0, 3)$ . Again, we must perform branch selection, and this time, we may choose to branch right, as shown in Figure 2(b).

Figure 1

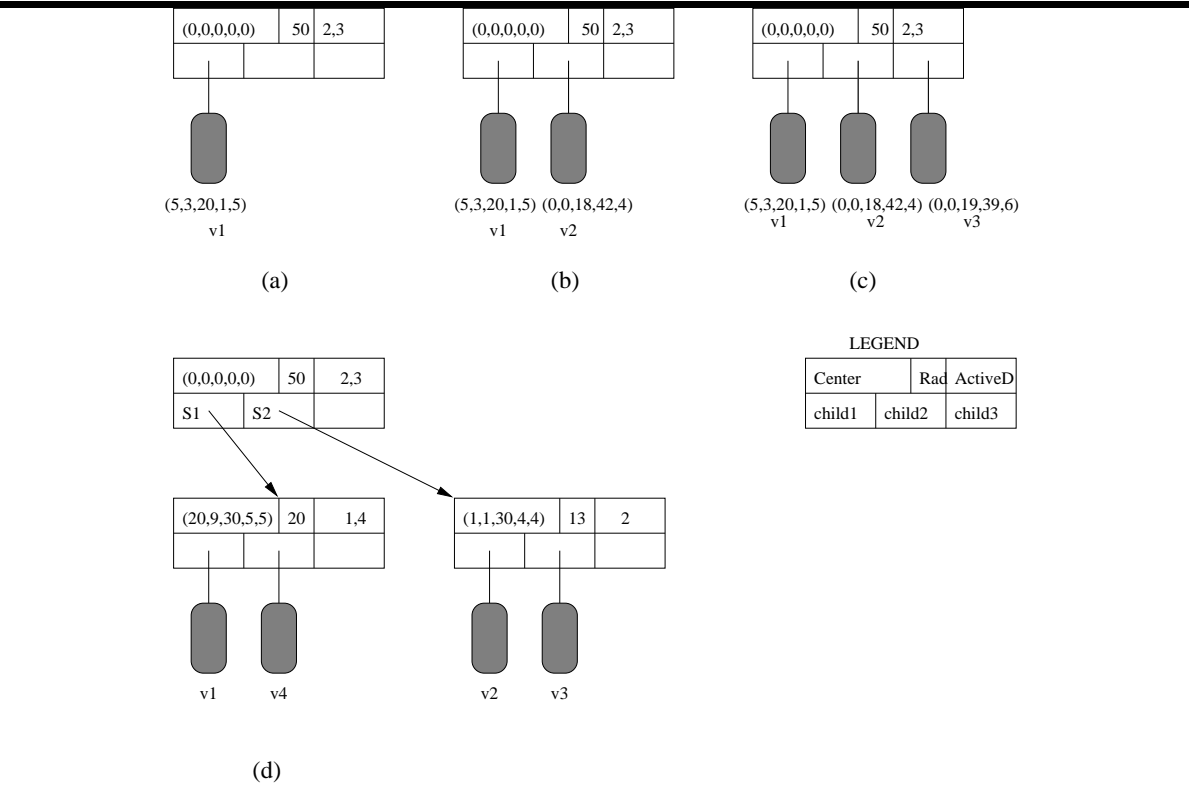
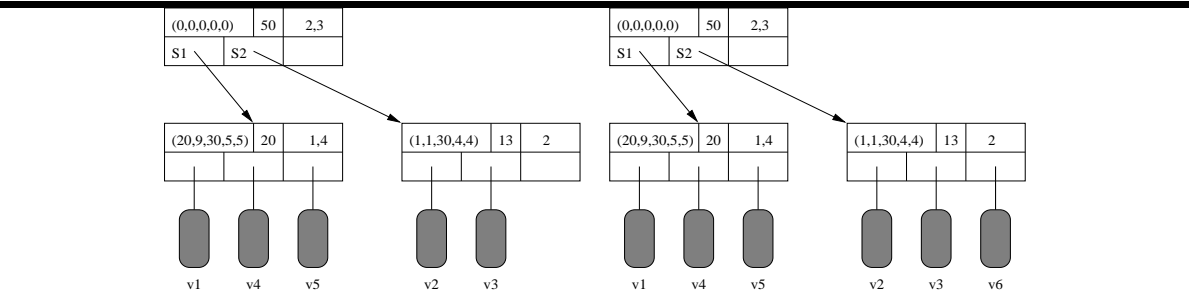


Figure 2



## Branch Selection

---

- Consider node  $N$  with  $1 \leq j \leq \text{NumChild}$  children, denoted  $N_1, \dots, N_{\text{NumChild}}$ .
- $\text{exp}_j(\mathbf{v})$  denotes the amount we must expand  $N_j.\text{Radius}$  so that  $\mathbf{v}$ 's active distance from  $N_j.\text{Center}$  falls within this radius.
- First, select all  $j$ 's such that  $\text{exp}_j(\mathbf{v})$  is minimized.  
EX: if we have nodes  $N_1, \dots, N_5$  with  $\text{exp}$  values 10, 40, 19, 10, 32 respectively, the two candidates selected for possible insertion are  $N_1$  and  $N_4$ . If a tie occurs, as in the above case, pick the node such that the distance from the center of that node to  $\mathbf{v}$  is minimized.

## Splitting

---

- When we attempt to insert a vector  $\mathbf{v}$  into a leaf node  $N$  that is already full, then we need to split the node.
- We must create subnodes  $N_1, N_2$ , and each vector in node  $N$  must fall into one of the regions represented by these two subnodes.
- Split vectors in leaf  $N$  into two groups  $(G_1, H_1)$ .
- We may be able to enclose all vectors in  $G_1$  within a region with center  $c_1$  and radius  $r_1$  and all vectors in  $H_1$  within a region with center  $c_2$  and radius  $r_2$ .
- Many such splits are possible in general.
- Split  $(G_1, H_1)$  is *finer than* split  $(G_2, H_2)$  iff the sums of the radii,  $(r_1 + r'_1)$  is smaller than the sum of the radii  $(r_2 + r'_2)$ .
- Still not enough to uniquely identify a “finest” split.
- If  $(G_1, H_1)$  and  $(G_2, H_2)$  are splits such that neither is finer than the other and no other split is finer than each of them, then we say that  $(G_1, H_1)$  is *more conservative* than  $(G_2, H_2)$  iff

$$r_1 + r'_1 - \text{act\_dist}(c_1, c'_1) \leq r_2 + r'_2 - \text{act\_dist}(c_2, c'_2).$$

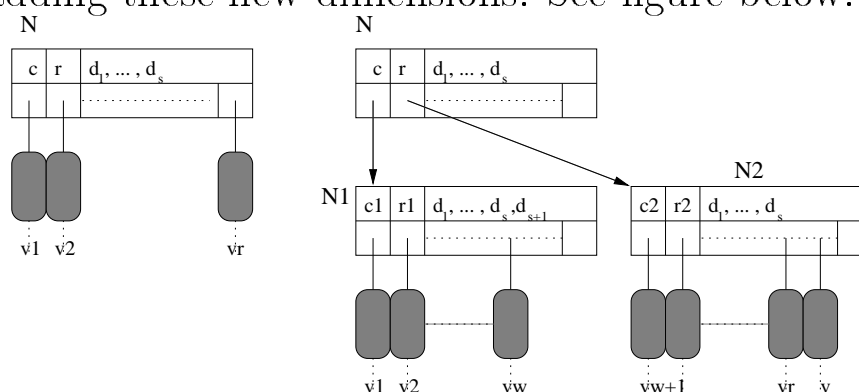
- Split  $(G, H)$  is the selected split iff:

1. there is no split  $(G', H')$  that is finer than  $(G, H)$  and

2. there is no split  $(G', H')$  that satisfies condition (1) (i.e. there exists no split  $(G^*, H^*)$  finer than  $(G', H')$ ) and that is more conservative than  $(G, H)$ .

## Telescoping

- Suppose  $N$  is the node into which we are to insert a vector  $\mathbf{v}$ .
- Insertion of  $\mathbf{v}$  may cause two types of changes to  $N$ : it may cause  $N$  to be *split* into two subnodes  $N_1, N_2$ , or it may “modify” the set of active dimensions of node  $N$  (e.g. if vector  $\mathbf{v}$  does not agree, on the active dimensions, with other vectors stored at node  $N$ ).
- When node  $N$  gets split into two sub-nodes  $N_1, N_2$ , the set of vectors at either node  $N_1$  or node  $N_2$  (but not both) must be a subset of the set of vectors at node  $N$  before the insertion.
- Suppose  $N_1$  has this property.
- The vectors in  $N_1$  may agree not only on the active dimensions of  $N$ , but on some other dimensions as well. In this case, we can *expand* the set of active dimensions of node  $N$ , by adding these new dimensions. See figure below.



- The other case when telescoping occurs is when a vector is added to a node,  $N$ , but no split occurs. If  $N$  originally

contained the vectors  $\mathbf{v}_1, \dots, \mathbf{v}_r$  and  $\mathbf{v}$  is the vector being added, even though vectors  $\mathbf{v}_1, \dots, \mathbf{v}_r$  originally agreed on the active dimensions  $d_1, \dots, d_s$  of node  $N$ , they only agree now on a subset (for example,  $d_2, \dots, d_s$ ) and hence, the set of active dimensions of node  $N$  must be *contracted* to reflect this fact.



## Searching in TV-Trees

---

**Algorithm 2** *Search*( $T, \mathbf{v}$ )

```
if Leaf( $T$ ) then Return ( $T.Center = \mathbf{v}$ ); Halt }  
else  
  { if  $\mathbf{v} \in Region(T)$  then  
    Return  $\bigvee_{i=1}^{NumChild} Search(T.Child[i], \mathbf{v})$   
  }  
end
```

## Nearest Neighbor Retrievals in TV-Trees

---

$$\begin{aligned} \min(N, \mathbf{v}) &= \begin{cases} 0 & \text{if } \mathbf{v} \in \text{Region}(N) \\ \text{act\_dist}(\mathbf{v}, N.\text{Center}) - N.\text{Radius} & \text{otherwise} \end{cases} \\ \max(N, \mathbf{v}) &= \text{act\_dist}(\mathbf{v}, N.\text{Center}) + N.\text{Radius}. \end{aligned}$$

- Maintain an array *SOL* of length *p*, i.e. with indices running 1 through *p*.
- The algorithm *NNSearch* uses a routine called *Insert* that takes as input, a vector *vec*, and an array *SOL* maintained in *non-descending order* of active distance from *vec*.
- *Insert* returns as output, the array *SOL* with *vec* inserted in it, at the right place, and with the *p*'th element of *SOL* eliminated.

## Nearest Neighbor Retrievals in TV-Trees (Contd.)

---

```

Algorithm 3 NNSearch( $T, \mathbf{v}, p$ )

  for  $i = 1$  to  $p$  do  $SOL[i] = \infty$ ;
  NNSearch1( $T, \mathbf{v}, p$ );

end ( $\star$  end of program NNSearch  $\star$ )

procedure NNSearch1( $T, \mathbf{v}, p$ );

  if Leaf( $T$ ) &  $\text{act\_dist}(T.val, \mathbf{v}) < SOL[p]$  then
    Insert  $T.val$  into  $SOL$ ;
  else
    {
      if Leaf( $T$ ) then  $r = 0$ ;
      else { Let  $N_1, \dots, N_r$  be the children of  $T$ ;
        Order the  $N_i$ 's in ascending order w.r.t. min( $N_i, \mathbf{v}$ );
        Let  $N_{\eta(1)}, \dots, N_{\eta(r)}$  be the resulting order;
      };
       $done = false$ ;  $i = 1$ ;
      while  $((i \leq r) \wedge \neg done)$  do
        {
          NNSearch( $N_{\eta(i)}, \mathbf{v}, p$ );
          if  $SOL[p] < \min(N_{\eta(i+1)}, \mathbf{v})$  then
             $done = true$ ;
             $i = i + 1$ ;
          }; ( $\star$  end of while  $\star$ )
        } ( $\star$  end of else  $\star$ )
    }
  Return  $SOL$ ;
end proc ( $\star$  end of subroutine NNSearch1  $\star$ )

```

## Other Retrieval Techniques: Inverted Indices

---

- A document record contains two fields – a **doc\_id** and a **postings\_list**.
- Postings list is a list of terms (or pointers to terms) that occur in the document. Sorted using a suitable relevance measure.
- A **term record** consists of two similar fields: a **term** field (string), and a **postings\_list**.
- Two hash tables are maintained: a **DocTable** and a **TermTable**. The **DocTable** is constructed by hashing on the **doc\_id** key, while the **TermTable** is obtained by hashing on the **term** key.
- To find all documents associated with a term, we merely return the postings list.
- Used in many commercial systems such as MEDLARS and DIALOG.

## Other Retrieval Techniques: Signature Files

---

- Associate a *signature*  $s_d$ , with each document  $d$ .
- A signature is a representation of an ordered list of terms that describe the document.
- The list of terms from which  $s_d$  is derived is obtained after performing a frequency analysis, stemming, and usage of stop lists.
- If signature list consists of the ordered list of words  $w_1, w_2, \dots, w_r$ .
- This means that word  $w_1$  is most important when describing the document, word  $w_2$  is second most important, and so forth.
- Signature of the document  $d$  is a *bit-representation* of this list, usually obtained by encoding the list after using hashing, and then superimposing a coding scheme.