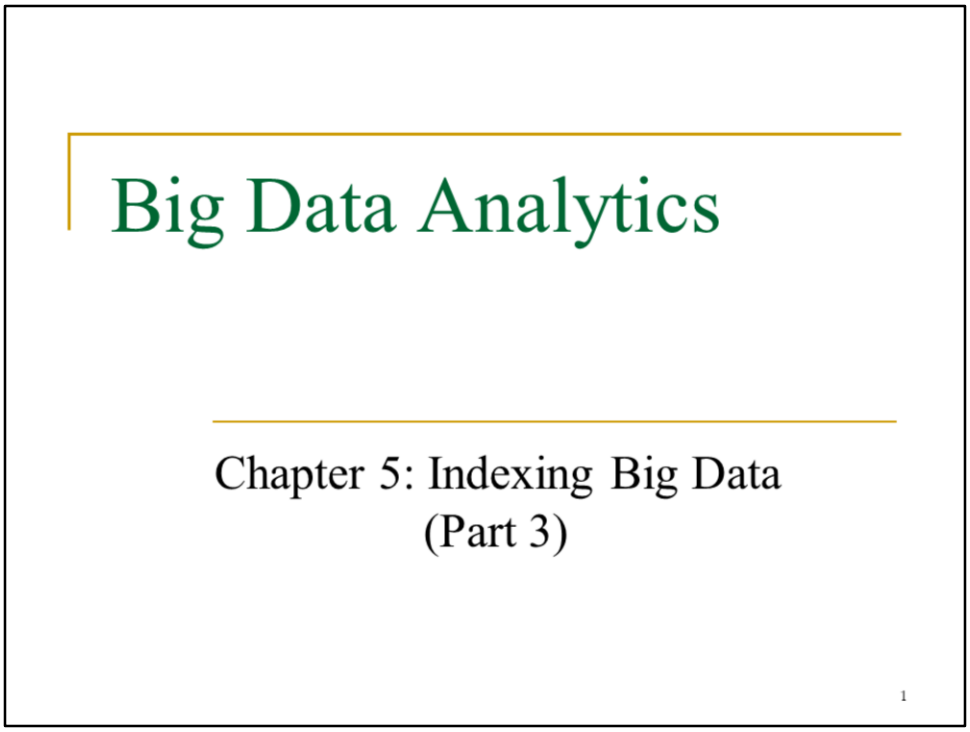
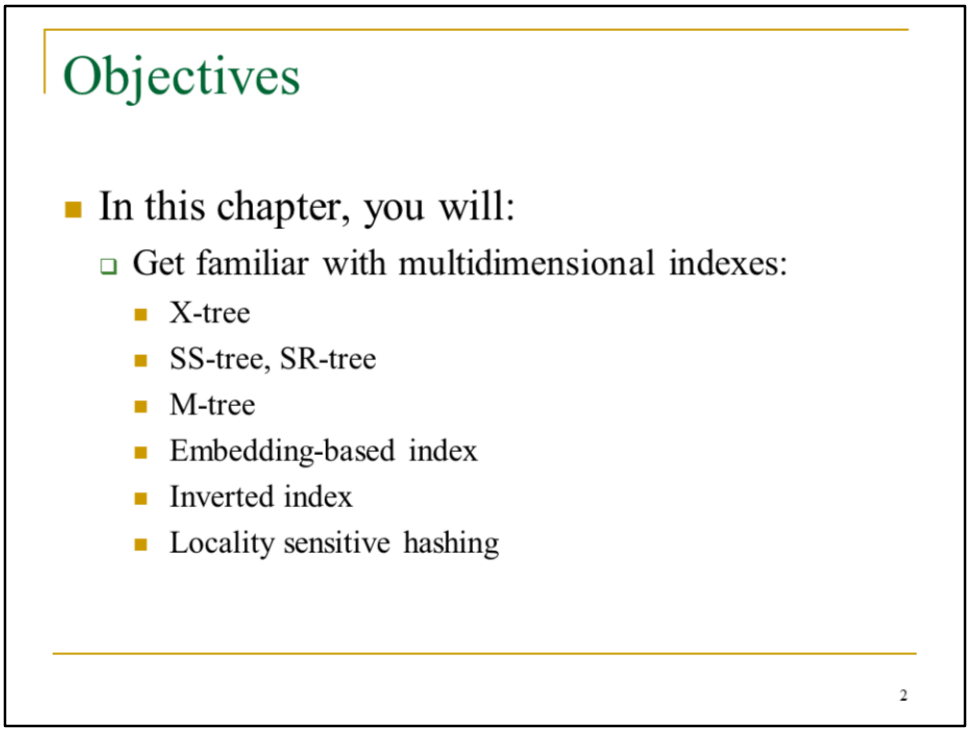


Chapter 5: Indexing Big Data (Part 3)



In this chapter, we start to discuss more indexes for high-dimensional data, including X-tree, SS-tree, SR-tree, M-tree, Embedding-based index, Inverted index, and Locality sensitive hashing.

Outline

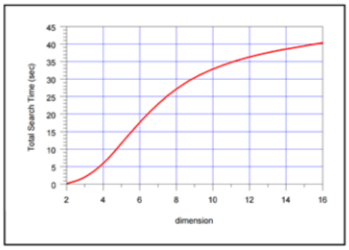
- X-tree
- SS-tree, SR-tree
- M-tree
- Embedding-based index
- Inverted index
- Locality sensitive hashing

3

Here is the outline of this chapter. We first will talk about the dimensionality curse for multidimensional indexes, and discuss more indexes for high dimensional data.

Disadvantage of R-Tree Family

- Dimensionality Curse
  - When the dimensionality, *d*, of the R-tree index (and its variants) is high (e.g., >16), the performance of R-tree degrades dramatically, which may be even worse than the *linear scan* method on the entire data set
  - The reasons are
    - In high dimensional space, there are many overlapping MBRs in the R-tree
    - The node capacity becomes smaller for high dimensional objects, which makes the tree higher



The performance of R-tree on the number of dimensions over real data

Source: S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An Index Structure for High-Dimensional Data. In *VLDB*, 1996.

4

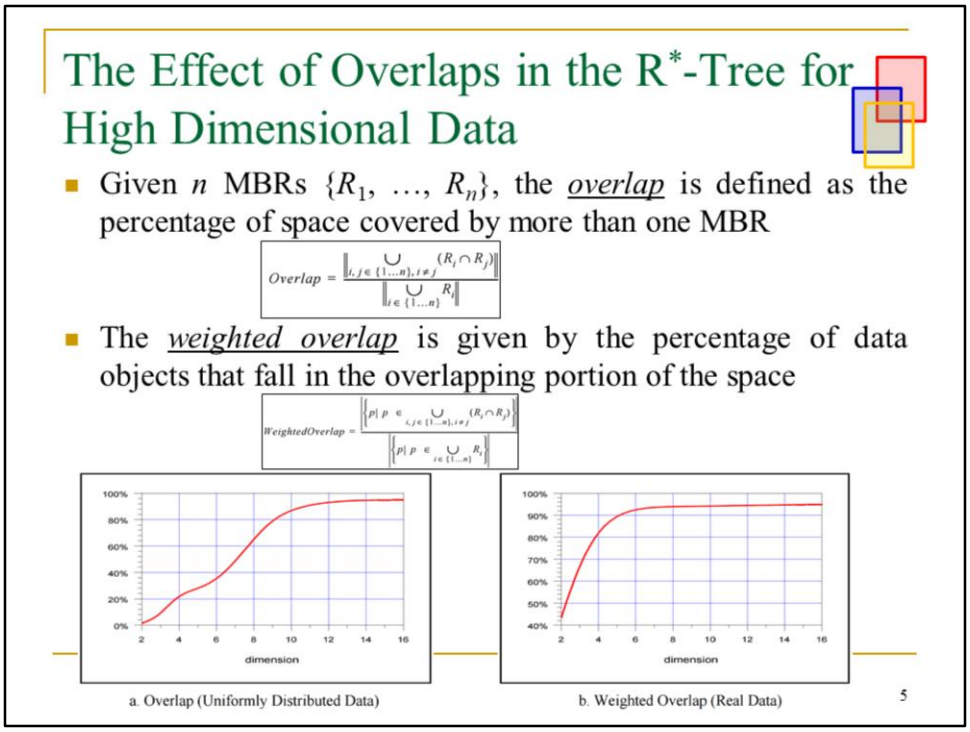
For many multidimensional indexes such as the R-tree family, there is a serious problem called dimensionality curse. In particular, when the dimensionality increases (e.g., more than 16), the performance of the indexes will degrade dramatically, which might be even worse than the linear scan method without any index.

The figure shows the performance of R-tree for different dimensionality. When the dimensionality increases to 16, the time cost is almost 1-2 orders of magnitude worse than that for dimensionality 2.

The dimensionality curse is due to the overlapping of MBRs in the R-tree for high dimensional data. Also, each node will retain fewer entries (child nodes or objects), since each object/MBR takes more space.

Reference:
"Curse of Dimensionality." *Wikipedia*, 7 Jan. 2019. *Wikipedia*, https://en.wikipedia.org/w/index.php?title=Curse_of_dimensionality&oldid=877317387.

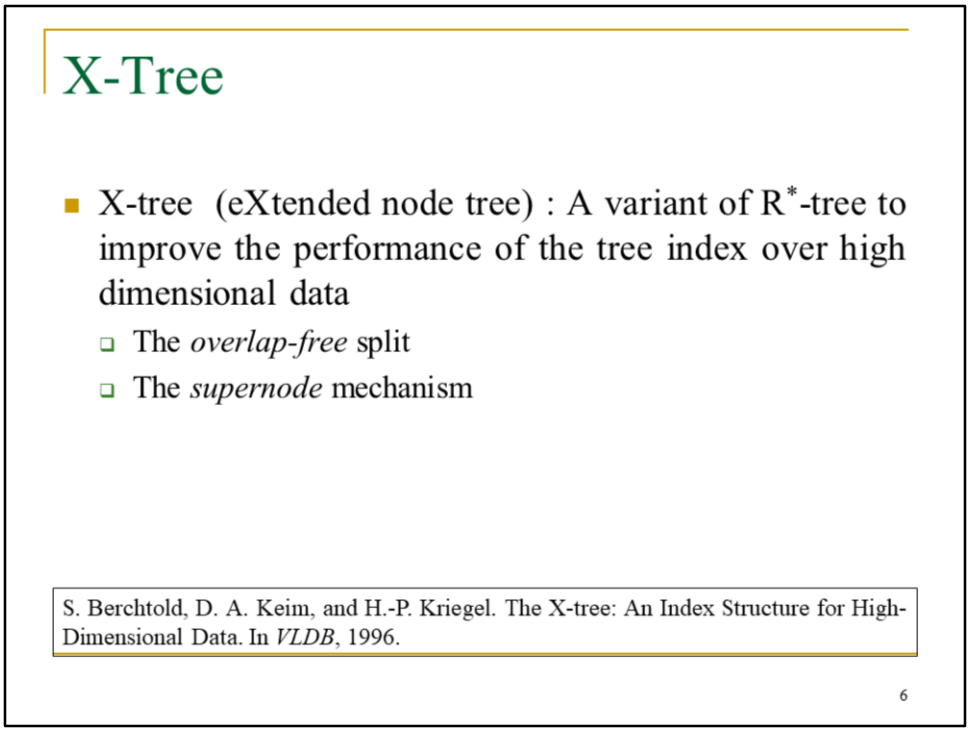
S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An Index Structure for High-Dimensional Data. In *VLDB*, 1996.



Given n MBRs, we define two terms, overlap and weighted overlap. The overlap is the percentage of space covered by more than one MBR, whereas the weighted overlap is the percentage of data objects that fall in the overlapping portion of the space. From figures, we can see that these two overlap measures greatly increase for high dimensionality. This is one of the reasons for the dimensionality curse.

Reference:

S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An Index Structure for High-Dimensional Data. In *VLDB*, 1996.

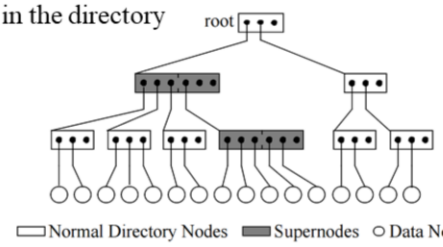


Inspired by this, X-tree index was proposed to index high dimensional data, which can achieve better performance than many multidimensional indexes. To overcome the shortcomings of R-tree index over high dimensional data, X-tree has the features of the overlap-free split and supernode mechanism.

Reference:
S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An Index Structure for High-Dimensional Data. In *VLDB*, 1996.

X-Tree Structure

- X-Tree structure
  - Data nodes
  - Normal directory nodes
  - Supernodes (with larger node sizes where necessary)
    - During insertions, supernodes are created to avoid overlap due to splits in the directory

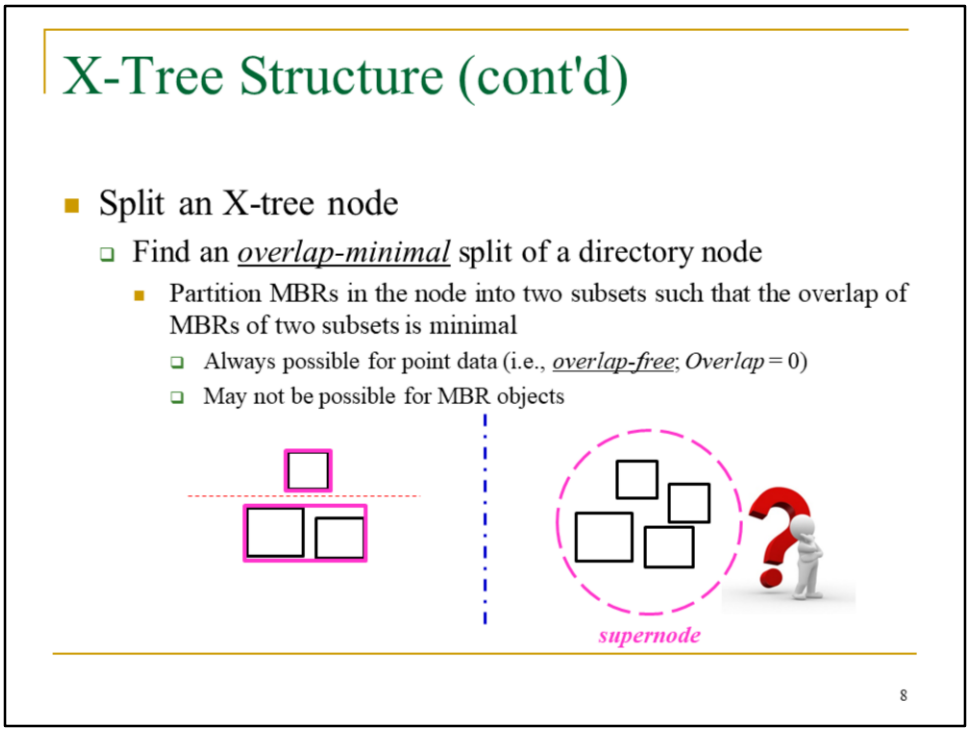


7

X-tree is a tree structure, consisting of data nodes (i.e., leaf nodes), normal directory nodes (i.e., non-leaf nodes), and supernodes (which can help avoid the overlaps).

Reference:

S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An Index Structure for High-Dimensional Data. In *VLDB*, 1996.



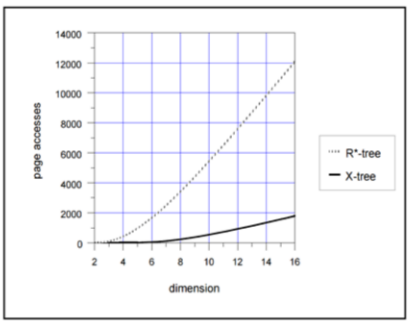
When a node is full, we may want to split the node into 2 groups so that the overlap of MBRs is minimum. However, it is not always the case that we can obtain 2 overlap-free nodes. For example, for the left figure, we can easily divide MBRs into 2 groups without any overlapping. In the right figure, it is not possible to divide the 4 MBRs into 2 groups such that the resulting MBRs of the two groups do not have any overlap. In this case, X-tree uses the mechanism of supernode, that is, it does not split this node, but keeps it as one single super node.

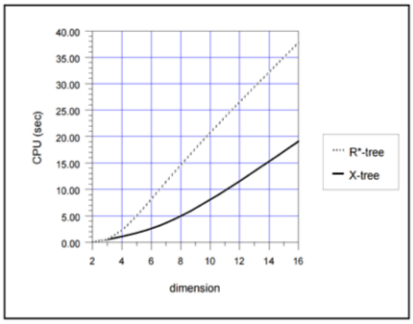
Reference:

S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An Index Structure for High-Dimensional Data. In *VLDB*, 1996.

Performance Comparisons Between R*-Tree and X-Tree

- Real Point Data (70 MBytes)





9

Here is the comparison between R*-tree and X-tree vs. dimensionality, in terms of I/O cost and CPU time. We can see that X-tree performs much better than R*-tree for high dimensionality like 16.

Reference:

S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An Index Structure for High-Dimensional Data. In *VLDB*, 1996.

Outline

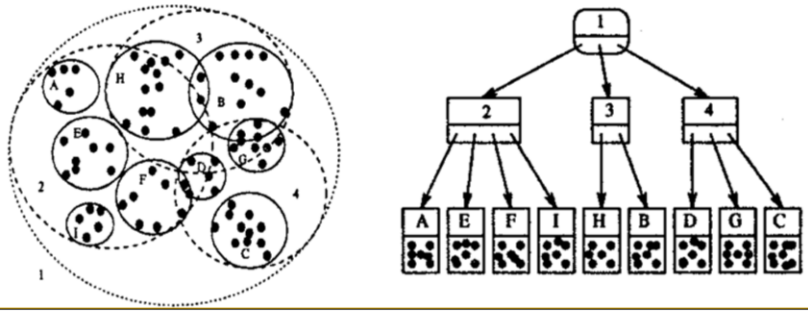
- X-tree
- SS-tree, SR-tree
- M-tree
- Embedding-based index
- Inverted index
- Locality sensitive hashing

10

Next, we discuss SS-tree and SR-tree.

SS-Tree

- The SS-tree is a height-balanced tree, where each node is a sphere
- The distance between two objects is given by a weighted Euclidean distance $D(\mathbf{x},\mathbf{y}) = \sqrt{(\mathbf{x}-\mathbf{y})^T \mathrm{diag}(\mathbf{w})(\mathbf{x}-\mathbf{y})}$



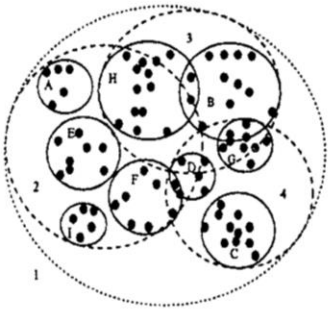
D. A. White and R. Jain. Similarity Indexing with the SS-tree. In *ICDE*, 1996.

The SS-tree is also a height-balanced tree, where each node is a sphere. The distance between two objects is defined by a weighted Euclidean distance.

Reference:
D. A. White and R. Jain. Similarity Indexing with the SS-tree. In *ICDE*, 1996.

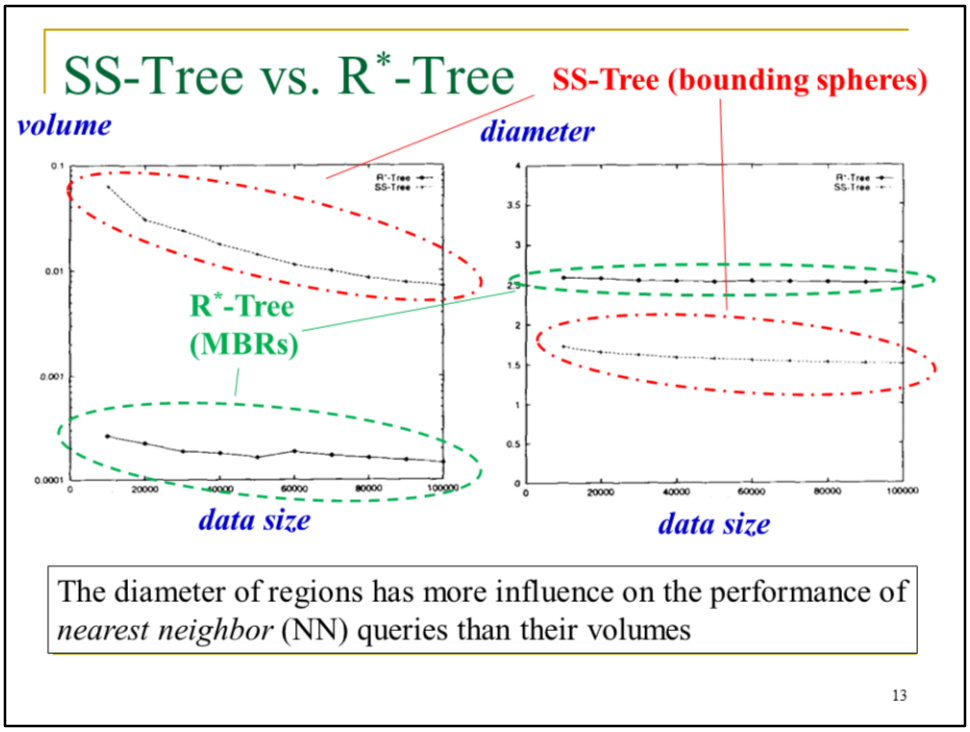
SS-Tree (cont'd)

- Each node in SS-tree is represented by a sphere, centered at a centroid of underlying objects and with a radius *r*



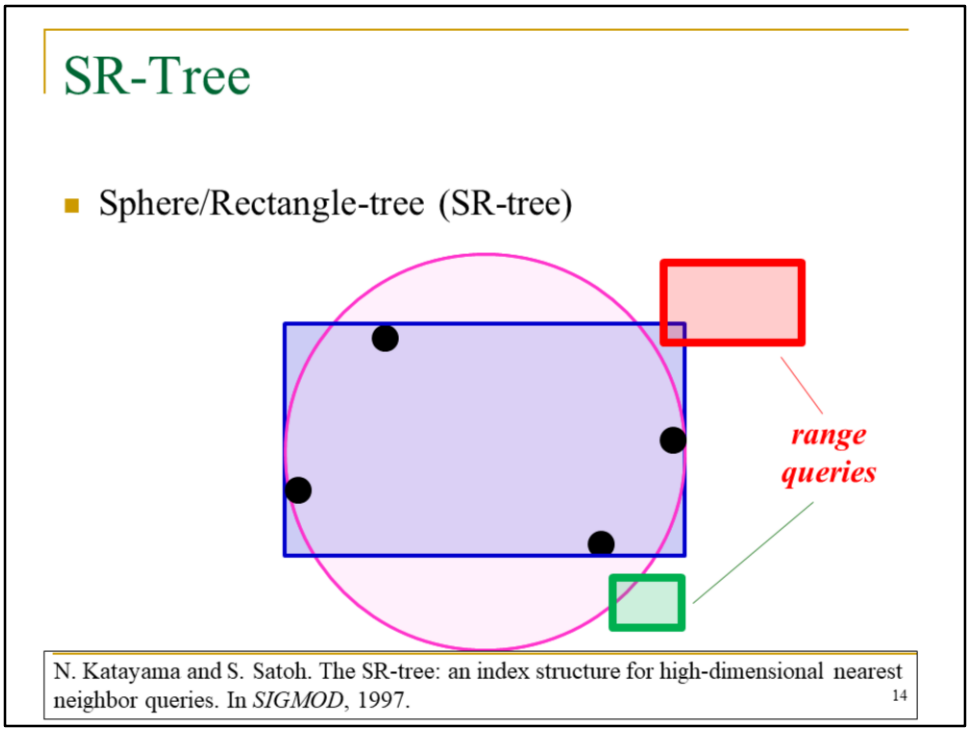
12

Each node consists of several objects or node entries, enclosed by a sphere centered at a centroid of objects under the node.



Although the volume of the sphere in SS-tree is much larger than that of R*-tree, the diameter of the sphere is smaller than that of R*-tree. The performance of nearest neighbor (NN) query is more influenced by the diameter than the volume. Thus, SS-tree can work well for NN queries over high dimensional data.

Reference:
N. Katayama and S. Satoh. The SR-tree: an index structure for high-dimensional nearest neighbor queries. In *SIGMOD*, 1997.



Therefore, the SR-tree is proposed to combine the strengths from SS-tree and R*-tree. As shown in the figure, assume that we use both rectangle and circle to minimally bound spatial objects in a node. Given a range query, if the query range only intersects with one of the representations (i.e., either MBR or circle), but not both, then this node can still be pruned (since objects lie in the overlapping region between MBR and circle).

Reference:

Katayama, N., & Satoh, S. (1997). The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. SIGMOD Conference.

SR-Tree (cont'd)

- Insert Algorithm – similar to the SS-tree
  - Update both bounding rectangles and spheres
  - The bounding sphere
    - Centroid of points, $x(x_1, x_2, \ldots, x_d)$

$$x_i = \frac{\sum_{k=1}^{n} C_k.x_i \times C_k.w}{\sum_{k=1}^{n} C_k.w}$$

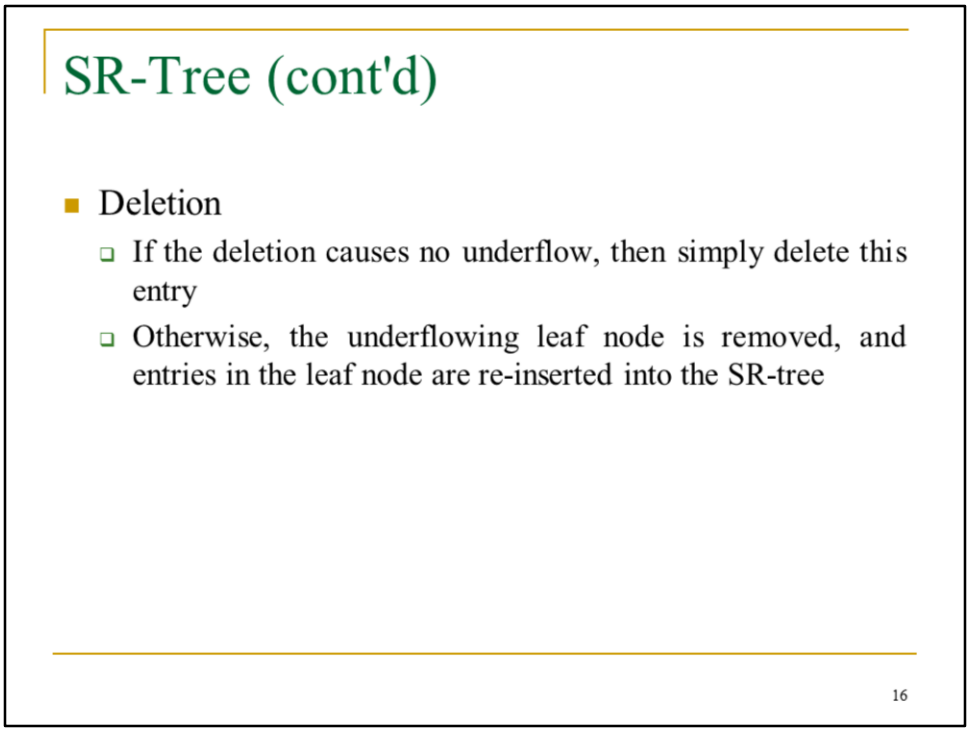
    - Radius r

$$\begin{aligned} r &= \min\ (d_s, d_r), \\ d_s &= \max_{1 \le k \le n} (\|\boldsymbol{x} - C_k.\boldsymbol{x}\| + C_k.r), \\ d_r &= \max_{1 \le k \le n} (MAXDIST(\boldsymbol{x}, C_k.\boldsymbol{R})). \end{aligned}$$

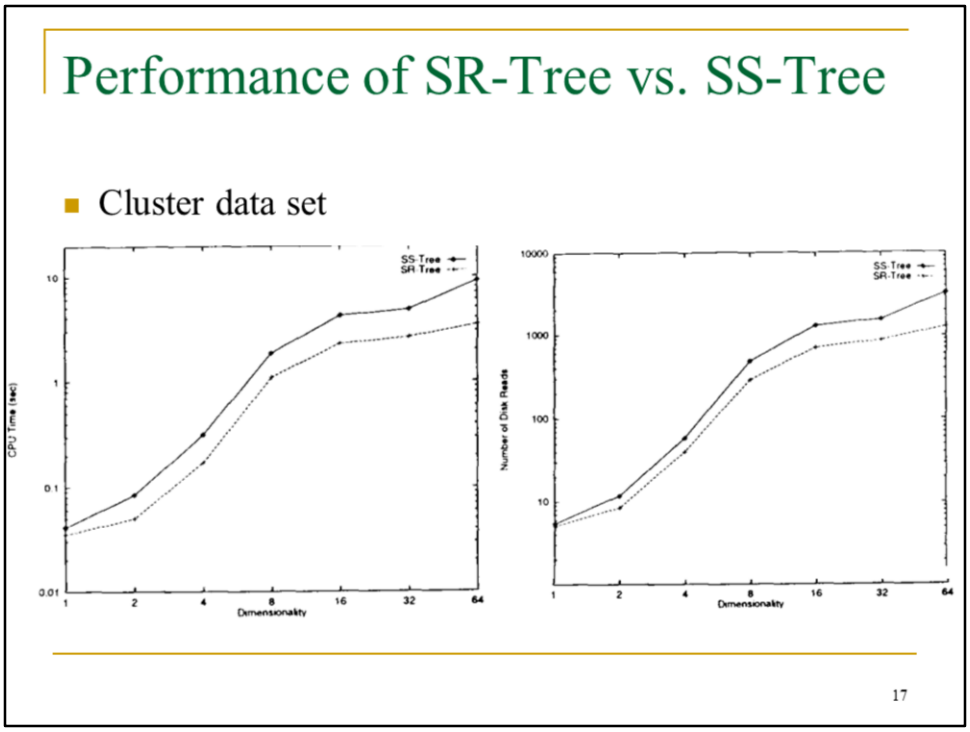
15

The insert algorithm is similar to the SS-tree. Both MBR and sphere should be updated.

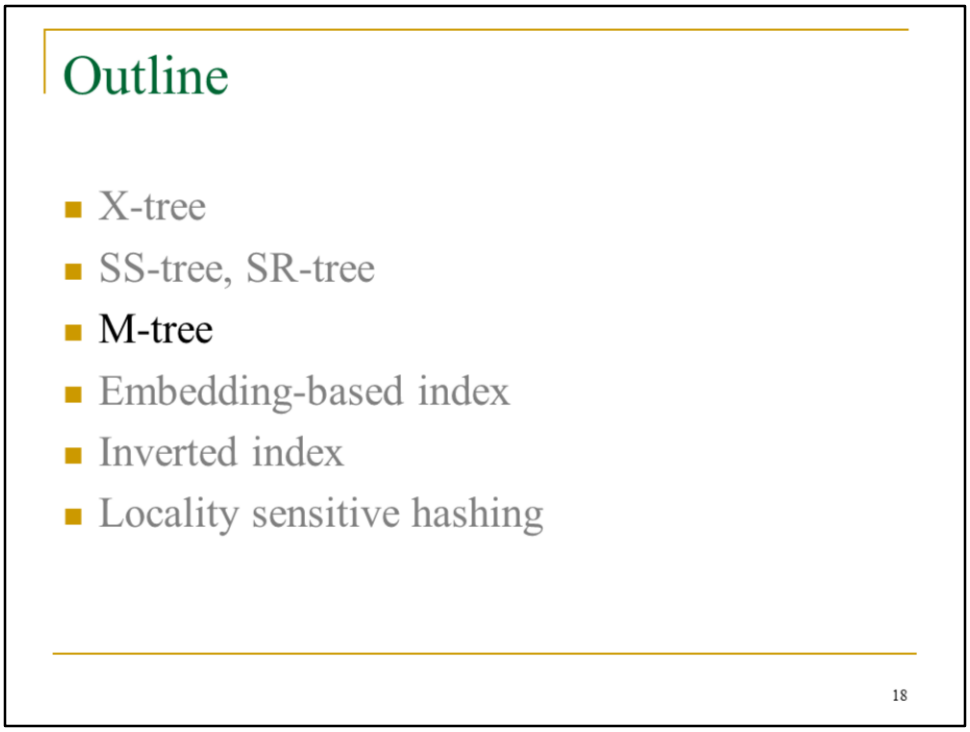
For the bounding sphere, here are the formulae of computing the centroid of multiple points and the radius.



The deletion algorithm also follows the style of re-insertion, similar to R*-tree.



Here are comparisons between SS-tree and SR-tree for high dimensionality. The SR-tree can outperform SS-tree for different dimensionality, in terms of CPU time and I/O cost.



So far we have talked about many indexes in the Euclidean space (i.e., the distance function between any two objects is a Euclidean distance function). We will next talk about the M-tree index in the metric space.

M-Tree (cont'd)

- M-tree: A height-balanced tree index in metric spaces
- Properties in metric spaces
  - Symmetry: $dist(x, y) = dist(y, x)$
  - Non-negativity: $dist(x, y) > 0$ $(x \neq y)$ and $dist(x, x) = 0$
  - Triangle inequality: $dist(x, y) + dist(y, z) \geq dist(x, z)$
- Euclidean distance is one type of metric-space distances
  - $dist(p, q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$

P. Ciaccia, M. Patella, and P. Zezula. M-tree An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB*, 1997.

19

The M-tree is a height-balanced tree designed for indexing objects in metric space. Euclidean space is just one special case of the metric space.

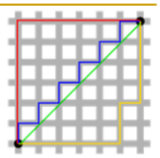
Metric space has 3 major properties, symmetry, non-negativity, and triangle inequality. For the symmetry, the distance from x to y is the same as that from y to x. For non-negativity, the distance between x and y is positive if x and y are two distinct objects, and the distance between x and itself is equal to 0. For the triangle inequality, the distance from x to y plus the distance from y to z is always greater than or equal to that from x to z.

Euclidean distance function is one of the metric distance functions.

Reference:

P. Ciaccia, M. Patella, and P. Zezula. M-tree An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB*, 1997.

Metric Spaces



- L_p-norm distance
  - L_1-norm (Manhattan distance)
  - L_2-norm (Euclidean distance)
  - L_∞-norm

$$\mathcal{L}_p(\vec{x} - \vec{y}) = \left(\sum_{i=1}^{L} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

$$\mathcal{L}_\infty(\vec{x} - \vec{y}) = \max_{i=1}^{L} |x_i - y_i|$$

- Edit distance
  - Given two strings a and b, **the edit distance**, $dist(a, b)$, is the minimum-weight series of edit operations that transforms a into b
    - Insertion, deletion, substitution
- The shortest path distance in graphs

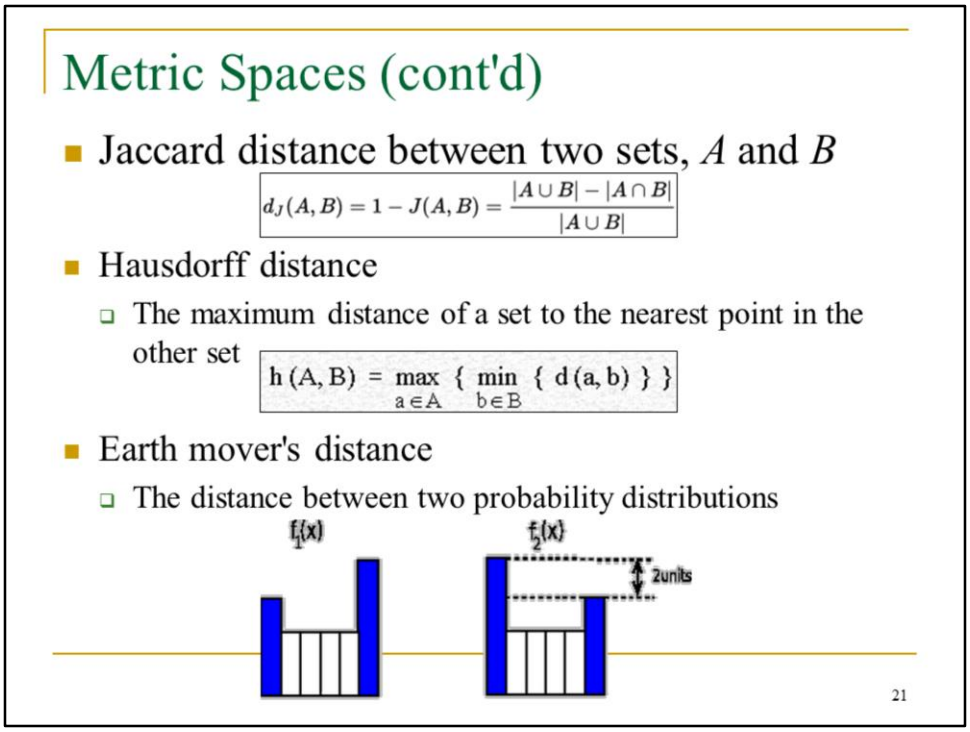
20

There are many other metric distance functions such as L_p-norm, including L_1-norm (or called Manhattan distance), L_2-norm (Euclidean distance), and L_\infty-norm, edit distance between two strings (via insertion, deletion, and substitution), and the shortest path distance in graphs.

References:

Byoung-Kee Yi and Christos Faloutsos. Fast Time Sequence Indexing for Arbitrary Lp Norms. In Proceedings of the 26th International Conference on Very Large Data Bases (VLDB), 2000.

“Edit Distance.” *Wikipedia*, 15 Jan. 2019. *Wikipedia*, https://en.wikipedia.org/w/index.php?title=Edit_distance&oldid=878601173.



Jaccard distance is also a metric distance function, which computes the distance between two sets A and B. Haustorff distance is given by the maximum distance of a set to the nearest point in the other set. Moreover, earth mover's distance is to compute the distance between two probability distributions. Intuitively, we can consider the distribution as earth, and the earth mover's distance gives the minimum efforts to convert from one distribution to another distribution.

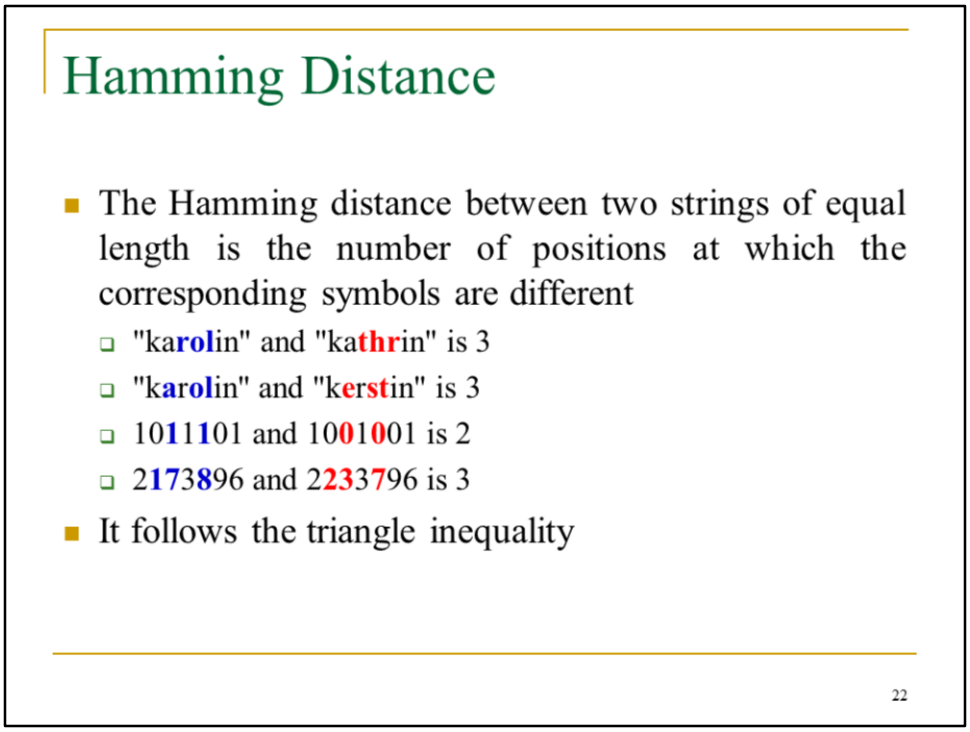
These distances are metric distance functions and follow the triangle inequality.

References:
"Jaccard Index." *Wikipedia*, 2 Dec. 2018. *Wikipedia*, https://en.wikipedia.org/w/index.php?title=Jaccard_index&oldid=871576429.

"Hausdorff Distance." *Wikipedia*, 10 Jan. 2019. *Wikipedia*, https://en.wikipedia.org/w/index.php?title=Hausdorff_distance&oldid=877749743.

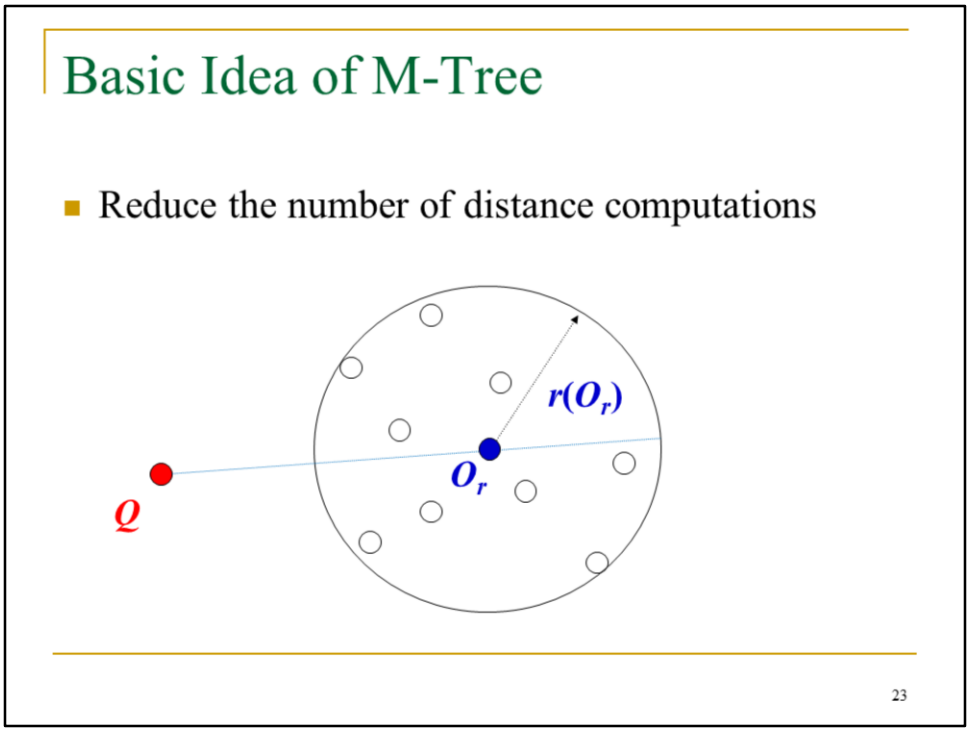
Hausdorff Distance. http://cgm.cs.mcgill.ca/~godfried/teaching/cg-projects/98/normand/main.html#whatis. Accessed 1 Feb. 2019.

Earth Mover's Distance -- EMD. http://vellum.cz/~mikc/oss-projects/CarRecognition/doc/dp/node29.html. Accessed 1 Feb. 2019.



Another metric distance is the hamming distance between two strings of the same length, which computes the percentage/number of different letters within two strings. For example, strings "karolin" and "kathrin" have 3 different letters. Their hamming distance is therefore equal to 3.

Reference:
"Hamming Distance." *Wikipedia*, 17 Jan. 2019. *Wikipedia*, https://en.wikipedia.org/w/index.php?title=Hamming_distance&oldid=878876975.

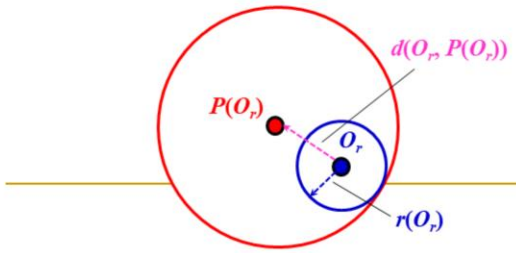


The basic idea of the M-tree is to utilize the triangle inequality to enable the pruning and query processing. Given a query point Q, and a node (centered at O_r with radius r(O_r)), the distance from Q to any object within the node can be bounded by [dist(Q, O_r) – r(O_r), dist(Q, O_r)+r(O_r)].

M-Tree Data Structure

- Leaf nodes
  - Store data objects
- Non-leaf nodes
  - Routing objects, O_r
  - Covering radii, $r(O_r)$
  - Distances, $d(O_r, P(O_r))$, from O_r to its parent $P(O_r)$
  - Pointers pointing to subtrees, $ptr(T(O_r))$

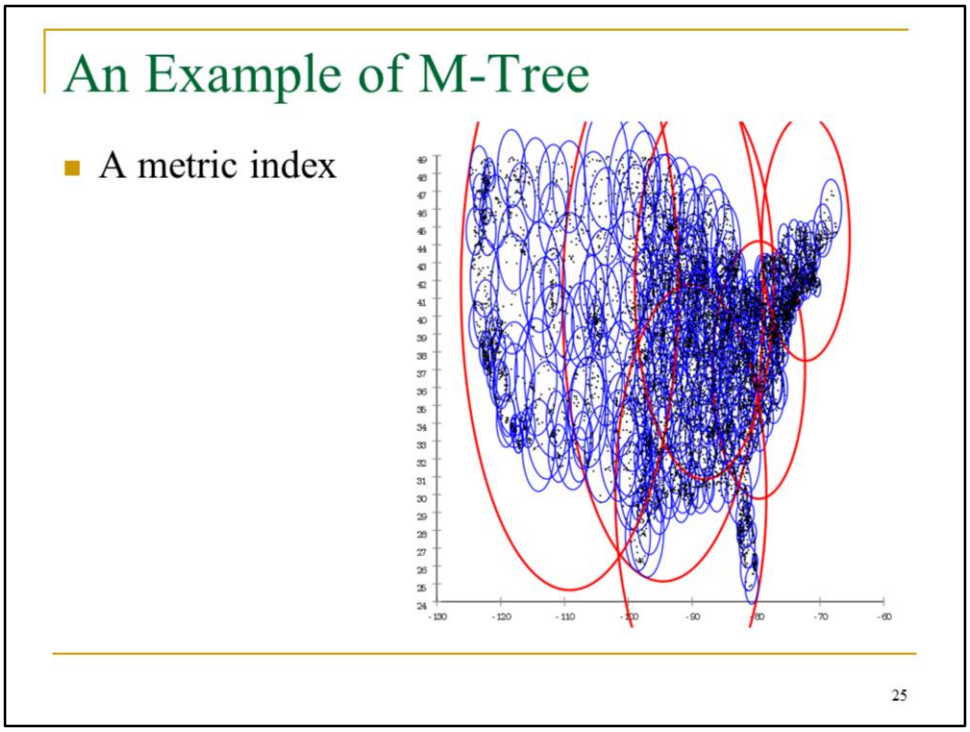
O_r	(feature value of the) routing object
$ptr(T(O_r))$	pointer to the root of $T(O_r)$
$r(O_r)$	covering radius of O_r
$d(O_r, P(O_r))$	distance of O_r from its parent



24

The M-tree is a tree structure, where each leaf node stores data objects and each non-leaf node stores entries, each with a routing object, a covering radius, distance to its parent and a pointer pointing to the subtree.

As shown in the figure, O_r is the routing object, and the circle centered at O_r with radius r(O_r) is a node entry. We also record the distance from O_r to the routing object P(O_r) of the parent node.



Here is an illustration of the M-tree index. The top node is the root of the M-tree. Each leaf or non-leaf node stores a number of objects or node entries, respectively.

Reference:

Chire. *Deutsch: De:M-Baum Der Durch Sequentielles Einfügen Mit Der MMRad-Split-Strategie Aufgebaut Wurde, Und Die Postleitzahlen Der USA Enthält.English: En:M-Tree Built by Repeated Insertion Using the MMRad Split Strategy and Storing All US Postal Codes.* 29 July 2014. Own work, *Wikimedia Commons*, https://commons.wikimedia.org/wiki/File:M-tree_built_with_MMRad_split.png.

Insert Algorithm for M-Tree

- Insert a new object O_n
  - Starting from root, find a branch to descend such that:
    - $d(O_r, O_n) \leq r(O_r)$;
  - In the case of multiple branches satisfying the inequality, select the one with O_r having the closest distance to O_n
  - If no routing objects satisfying $d(O_r, O_n) \leq r(O_r)$ exist, the choice is to minimize the increase of the covering radius $d(O_r, O_n) - r(O_r)$
  - When inserting O_n into a full leaf node, we split the overflow node into two nodes

26

To construct the M-tree, we can use the insert algorithm to insert objects one by one, which is quite similar to R*-tree. To insert a new object O_n, we can start from the root to find the branch to descend. We choose a branch if the new object O_n is inside a node (e.g., centered at O_r with radius r(O_r)). If there are multiple branches satisfying the condition, we select the one with routing object O_r closest to O_n.

If there are no nodes containing the new object O_n, we will select a node entry such that the increase of the covering radius is minimized. In the case that a leaf node to be inserted is full, we will split the overflow node into two.

Split Policies for M-Tree

- Goals
  - Minimizing "volume"
  - Minimizing "overlap"
- Policies
  - m_RAD – minimize (sum of) RADii, $r(O_{p1}) + r(O_{p2})$,
  - mM_RAD – minimize the maximum of the two radii, $\max\{r(O_{p1}), r(O_{p2})\}$
  - M_LB_DIST – minimize the Maximum Lower Bound on DISTance (i.e., O_{p2} is the farthest object from O_{p1})
  - RANDOM – randomly select routing objects
  - SAMPLING – select routing objects from s sample objects (s>1)

27

The splitting policy is to minimize both volume and overlap. For the details, there are different modes, for example, minimize the sum of radii from the two split nodes, or the maximum of radii from the two split nodes, and so on. For the detailed policies, you can check the first paper about the M-tree.

Distribution of Entries Between Two Split Nodes

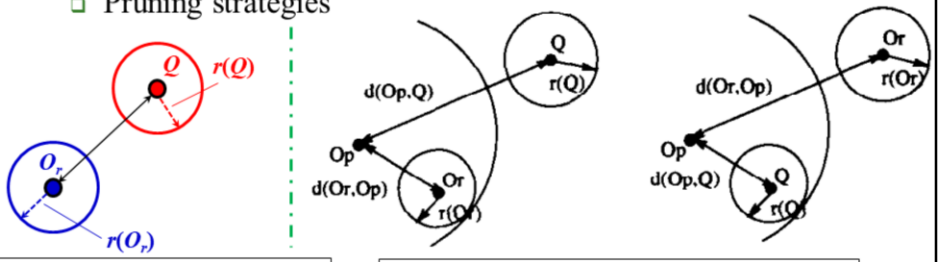
- Balanced distribution:
  - Compute $d(O_j, O_{p1})$ and $d(O_j, O_{p2})$ for all $O_j \in N$. Repeat until N is empty:
    - Assign to N_1 the nearest neighbor of O_{p1} in N and remove it from N;
    - Assign to N_2 the nearest neighbor of O_{p2} in N and remove it from N.

28

To distribute entries between two split nodes, we in turn find the nearest neighbor of each routing object for the two split nodes, and assign the nearest neighbor to the corresponding split node.

Range Queries Over M-Tree

- Given a query object Q, and a query radius $r(Q)$, a *range query* is to find all objects O_j such that $d(O_j, Q) \leq r(Q)$
  - Pruning strategies



If $d(O_r, Q) > r(Q) + r(O_r)$, then objects in node O_r can be safely pruned

If $|d(O_p, Q)-d(O_r, O_p)|>r(Q)+r(O_r)$, then $d(O_r, Q) > r(Q) + r(O_r)$.

29

With the M-tree, we can issue a range query to find all objects within r(Q)-distance away from the query point Q. Given an M-tree node centered at O_r with radius r(O_r), as shown in the left figure, this node does not need to be accessed, if the distance from O_r to Q is greater than the sum of radii r(O_r) and r(Q).

Since it is not efficient to online compute the distance from Q to every routing object, instead, we can use offline pre-computed distance d(O_r, O_p) w.r.t. pivot O_p, and apply the triangle inequality to derive the pruning condition, as shown in the right figure. This is also one of your homework to prove the correctness of the pruning condition.

Outline

- X-tree
- SS-tree, SR-tree
- M-tree
- Embedding-based index
- Inverted index
- Locality sensitive hashing

30

We next talk about the embedding-based index.

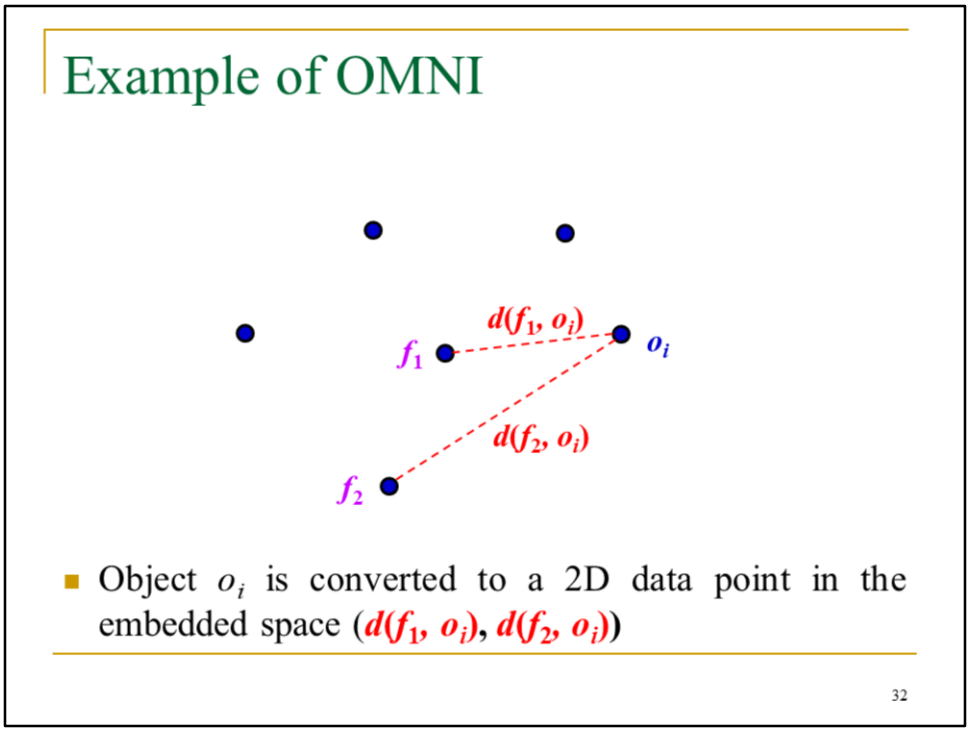
OMNI Family

- All-purpose access methods for similarity search in *metric* spaces
- Basic idea
  - Select a set, F, of objects as ***foci*** (or called ***pivots*** or ***reference points***), $f_1, f_2, \ldots, f_l$
  - Compute distances, $d(f_k, o_i)$, of all other objects, o_i, from this set F
  - Convert each object to a new l-dimensional data space, and conduct the search in the new space

R.F.S. Filho, A. Traina, C. Traina, and C. Faloutsos. Similarity search without tears: the OMNI-family of all-purpose access methods. In *ICDE*, 2001.

31

We will discuss all-purpose access methods for similarity search in metric spaces. The basic idea is to select l objects as foci (or called pivots or reference points). Then, each object is converted into an l-dimensional embedded space, where the k-th attribute is given by the distance from this object o_i to the k-th pivot.



As an example, in the figure, we have two foci f_1 and f_2. Object o_i is converted into a 2D embedded space with 2 attributes d(f_1, o_i) and d(f_2, o_i). This way, each object in the original data space is converted into a point in a 2D embedded space.

OMNI - Terms

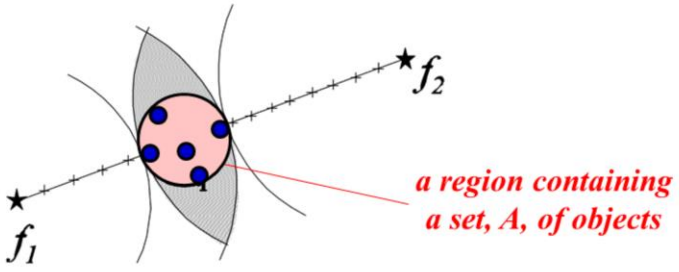
- Omni-foci base
  - A set, F, of foci $f_1, f_2, \ldots, f_l$
- Omni-coordinates
  - Converted coordinates, $d(f_k, o_i)$, of objects in the embedded space
- Minimum Bounding Omni Region (mbOr)
  - Each focus defines a metric subspace "ring"
  - The mbOr is a bit like "minimum bounding rectangle" of a set, A, of objects in the embedded space

33

Here are some terms related to OMNI. OMNI-foci base is a set of foci. OMNI-coordinates are the converted attributes of objects in the embedded space. Minimum bounding OMNI region is like MBR for R*-tree. The MBR in the embedded space has the shape of rings in the original data space.

OMNI – Example of Minimum Bounding Omni Region

- Minimum Bounding Omni Region (mbOr)
  - The intersection of the metric intervals, I_i, on the i-th dimension of the embedded space (w.r.t. focus f_i)
  - $I_i=[\min_{\forall oj \in A} \{d(o_j,f_i)\}, \max_{\forall oj \in A} \{d(o_j,f_i)\}]$

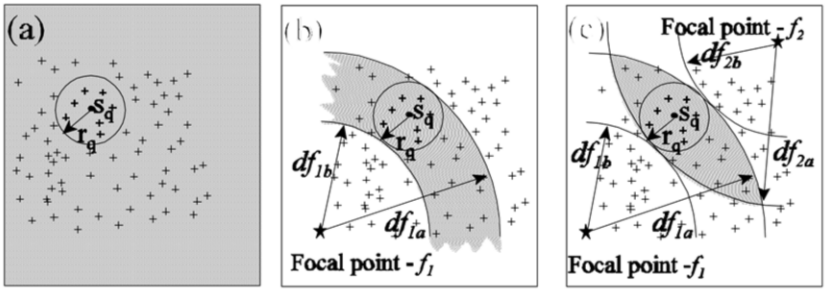


34

As shown in the figure, we have 2 foci f_1 and f_2, given a node containing objects, the distances from these objects to each focus are bounded by minimum and maximum possible distances from focus f_1 or f_2 to this set of objects. Intuitively, in the embedded space, these bounds w.r.t. two foci (dimensions) are corresponding to an MBR.

Range Queries over OMNI

- Given a query point S_q, and a query radius r_q
  - Retrieve all objects with r_q distances from S_q



The triangle inequality in metric spaces!

35

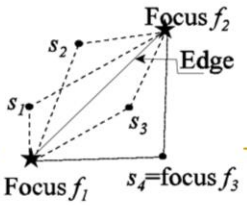
A range query retrieves all objects within a region centered at a query point S_q with a radius r_q. If there is no focus, then we have to access all objects in the data space as shown in Figure (a). If there is one focus, we only need to access the shaded ring region, as shown in Figure (b). Similarly, if there are two foci, we only need to access the shaded intersection region between two rings w.r.t. foci f_1 and f_2. Intuitively, with more foci, we can achieve higher pruning power, and need to access fewer candidates of the range query.

Research Problems with OMNI

- How to select foci?
  - HF-algorithm (finding foci near hull of the data set)
    - Randomly choose an object s_1
    - Find the farthest object f_1 from s_1 and set it as the first focus
    - Find the farthest object f_2 from f_1 and set it as the second focus
    - The next focus will be the one, s_i, that has the ***most similar distances*** *to* the previously chosen foci
      - Formally, minimizing

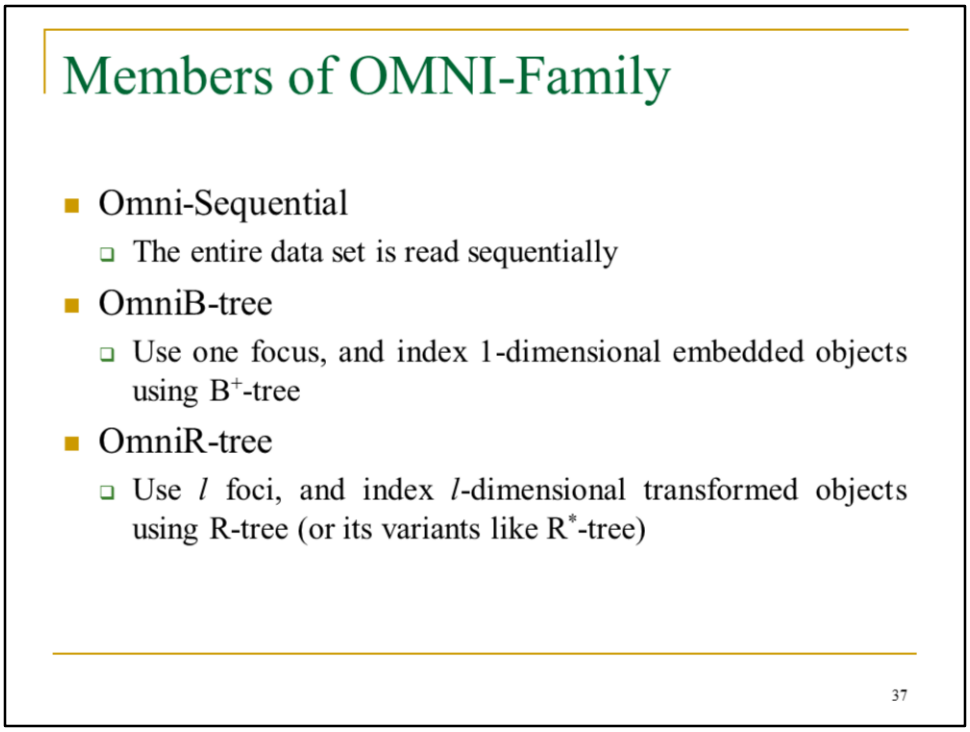
$$error_i = \sum_{k}^{k \text{ is focus}} |edge - d(f_k, s_i)|$$

$$edge = d(f_1, f_2)$$



36

One important problem for building an embedded-based index is how to select good foci. There is an HF-algorithm, which initially selects a random object s_1, and finds the farthest object f_1 of this object as the first focus. Next, we find the farthest object f_2 of f_1 as the second focus. Then, the next focus will be the one with the most similar distances to the previous foci. Intuitively, we want to have the next focus far from the previous ones so that the pruning power becomes higher.



We can consider 3 different embedded-based indexes in the OMNI-family.

If we do not use any indexes to organize objects in the embedded space, then this is called OMNI-sequential, in which all objects in the embedded data file should be sequentially accessed.

If we use only one focus to perform the embedding, then the converted 1-dimensional embedded objects are indexed by a B+-tree. The resulting index is called OMNIB-tree.

In the general case, if we use l foci, then we can use R*-tree to index the embedded l-dimensional objects. This index is called OMNIR-tree.

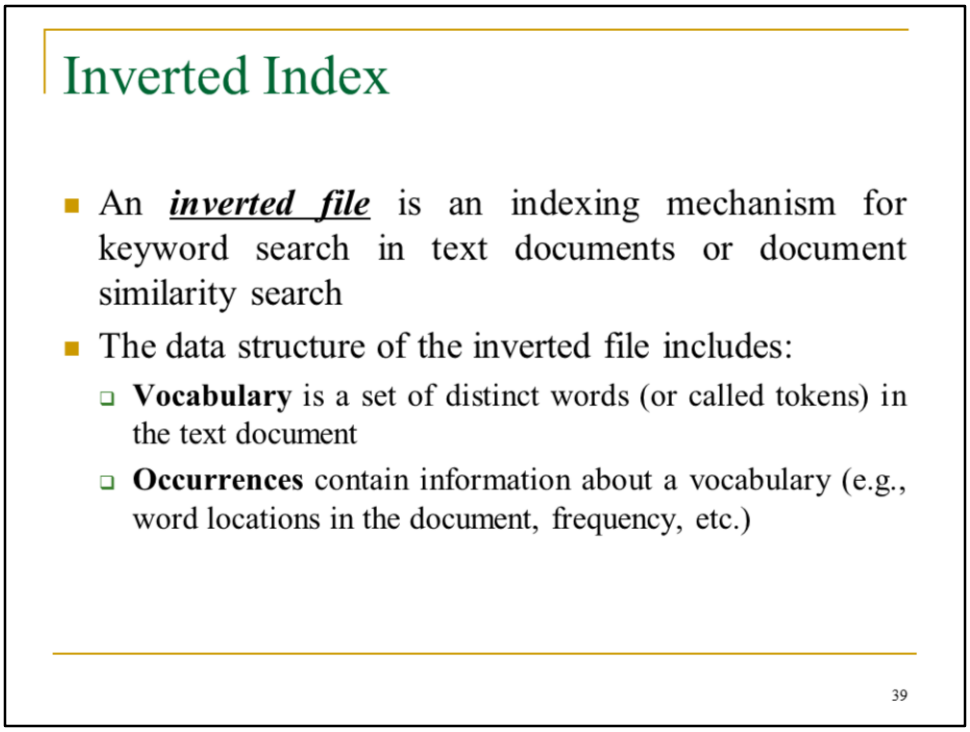
In fact, there are some other research topics such as how many foci to select. One possible direction is to obtain the intrinsic dimensionality of the data set, and use the number of foci/pivots the same as the intrinsic dimension.

Outline

- X-tree
- SS-tree, SR-tree
- M-tree
- Embedding-based index
- Inverted index
- Locality sensitive hashing

38

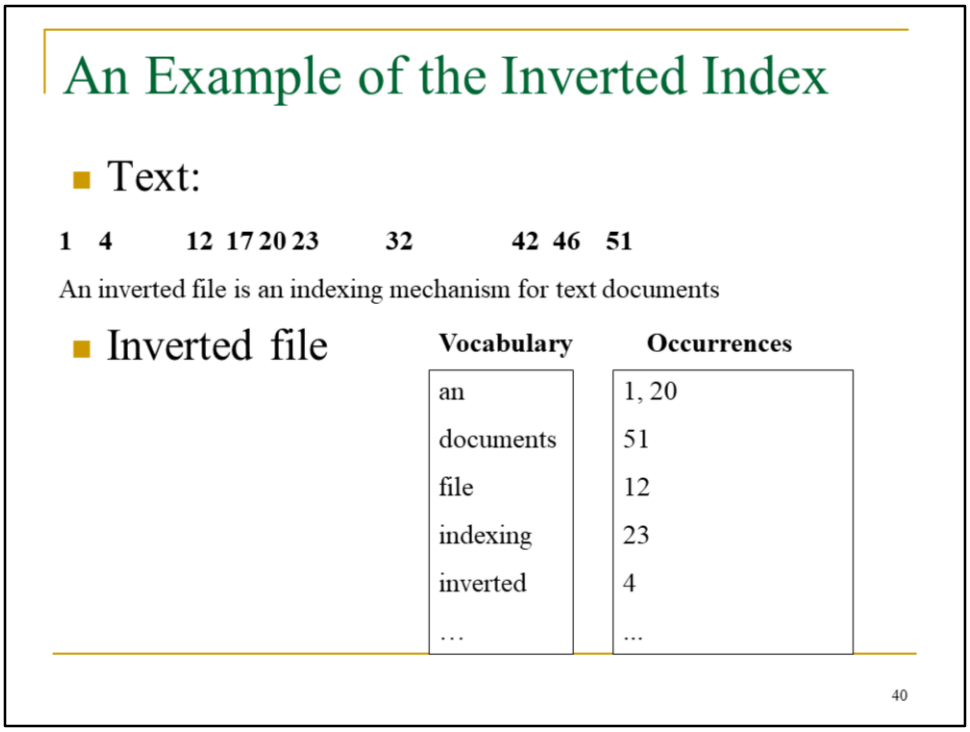
We next talk about the inverted index on text documents.



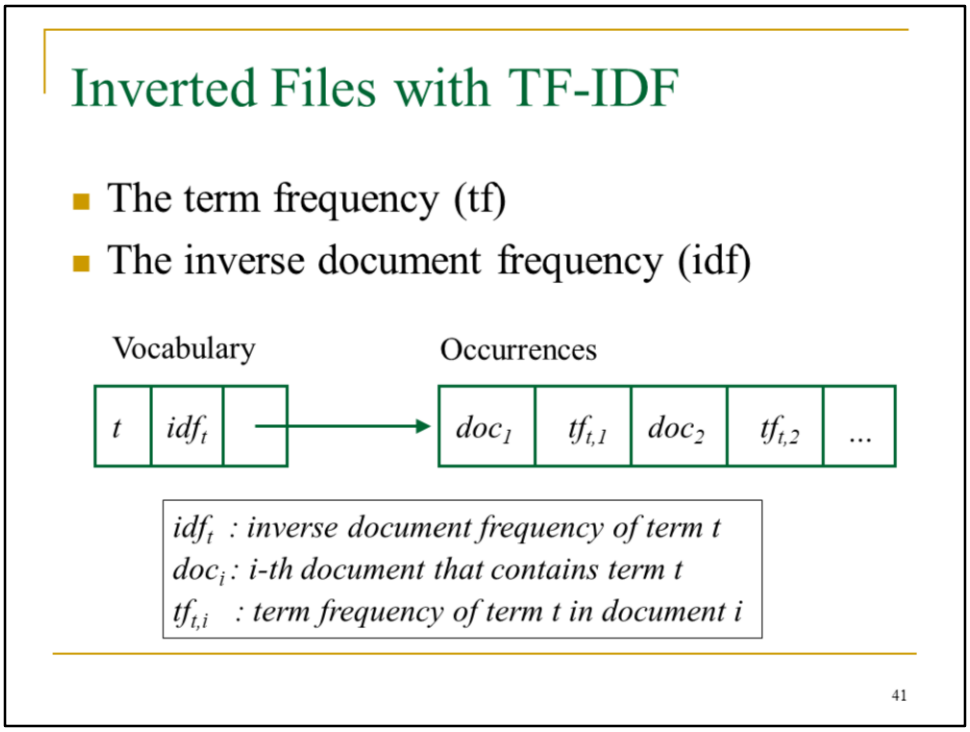
An inverted file is an indexing mechanism for text documents to enable fast keyword search and document similarity search. The inverted file has two main data structures, vocabulary and occurrences.

Vocabulary is a set of distinct words (a.k.a. tokens) in the text. Occurrences include statistics of each word that appears in documents (such as word location in the document, frequency in documents, and so on).

Reference:
"Inverted Index." *Wikipedia*, 11 Sept. 2018. *Wikipedia*, https://en.wikipedia.org/w/index.php?title=Inverted_index&oldid=859071935.



Here is an example of a text sentence. We extract vocabulary from this sentence, which contains words such as an, documents, file, indexing, inverted, and so on. We also record the occurrences of each word in this sentence. For example, “documents” appears at the location 51, and “an” appears twice at the locations 1 and 20.



In addition to occurrences of each word, we also store some other statistics that may facilitate document/keyword similarity search. For example, tf and idf are two important measures about term frequency and inverse document frequency.

Reference:

V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In VLDB, 2002.

Inverted Files with TF-IDF (cont'd)

- Term frequency, tf(t, d)
  - The number of times that term t occurs in document d

weighting scheme	TF weight
binary	$0, 1$
raw frequency	$f_{t,d}$
log normalization	$1 + \log(f_{t,d})$
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$
double normalization K	$K + (1 - K)\frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

42

For the term frequency, it is the number of times that a term appears in a document.

There are many other variants of the functions to define the term frequency, such as binary, raw frequency, log normalization, and so on.

Reference:
"Tf–Idf." *Wikipedia*, 17 Jan. 2019. *Wikipedia*, https://en.wikipedia.org/w/index.php?title=Tf%E2%80%93idf&oldid=878839239.

Inverted Files with TF-IDF (cont'd)

- Inverse document frequency, idf(t, D)

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

weighting scheme	IDF weight ($n_t = \|\{d \in D : t \in d\}\|$)
unary	1
inverse document frequency	$\log \frac{N}{n_t}$
inverse document frequency smooth	$\log\left(\frac{N}{1+n_t}\right)$
inverse document frequency max	$\log\left(\frac{\max_{\{t' \in d\}} n_{t'}}{1+n_t}\right)$
probabilistic inverse document frequency	$\log \frac{N-n_t}{n_t}$

43

Similarly, we also have different definitions about inverse document frequency (i.e., the inverse of the number of documents containing the term), which has been used in the information retrieval (IR) area.

Reference:
"Tf–Idf." *Wikipedia*, 17 Jan. 2019. *Wikipedia*, https://en.wikipedia.org/w/index.php?title=Tf%E2%80%93idf&oldid=878839239.

Inverted Files with TF-IDF (cont'd)

- Term frequency - Inverse document frequency
  - Reflect how important a word is to a document in a collection or corpus

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D)$$

- Higher tf-idf measure indicates more importance of term t

44

We usually define the similarity between two text documents by TF and IDF, namely tf-idf, which multiplies tf with idf for a term t. Intuitively, tf-idf measures the importance of the term (or word) in a document.

Reference:
"Tf–Idf." *Wikipedia*, 17 Jan. 2019. *Wikipedia*, https://en.wikipedia.org/w/index.php?title=Tf%E2%80%93idf&oldid=878839239.

Score for a Document

- Given a query document q and a data document d
- The similarity score between q and d can be defined by:
  - score $(q, d) = \Sigma_{\forall t \in q \wedge d} \text{tfidf}(t, d, D)$
  - where t is a term in both documents q and d

45

Then, for every term within two documents, we have the similarity score between a query document q and a data document d in this formula. Intuitively, if two documents have many common terms with high importance, then these two documents are more likely to be similar.

The Search Algorithm

- The search algorithm on an inverted index:
  - Search the vocabulary so that the words present in the query are found
  - Retrieve occurrences of all query words
  - Compute the document similarity with respect to the occurrences, in order to find search results

46

To search via the inverted index, we first use vocabulary to find words in the query. Then, we retrieve lists of occurrences for all query keywords. Finally, we manipulate occurrences in order to answer the search query.

The Search Algorithm (cont'd)

- To search the vocabulary, we can use index structures
  - Hashing: $O(1)$ cost
  - Tries: $O(c)$ cost, c = the number of letters in the word
  - B+-trees: $O(\log v)$ cost, v = the total number of keywords
- To search the vocabulary without indexes, we can store words in lexicographical order
  - No space cost for indexes
  - $O(log\ v)$ binary search cost

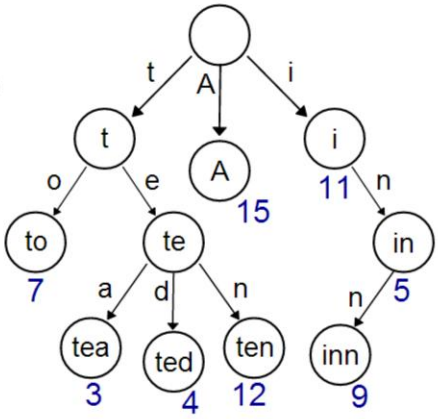
47

To search vocabulary, we can use a hash index to quickly look up a particular keyword with O(1) cost. Or we can use tries with O(c) cost, where c is the number of letters in a word. Another way is to use B+-tree, with O(log v) cost, where v is the number of keywords in the B+-tree.

Without indexing, an alternative is to store words in alphabetical order and perform the binary search with O(log v) cost.

Trie

- Trie (or prefix tree)
  - An index for string data



https://en.wikipedia.org/wiki/Trie

48

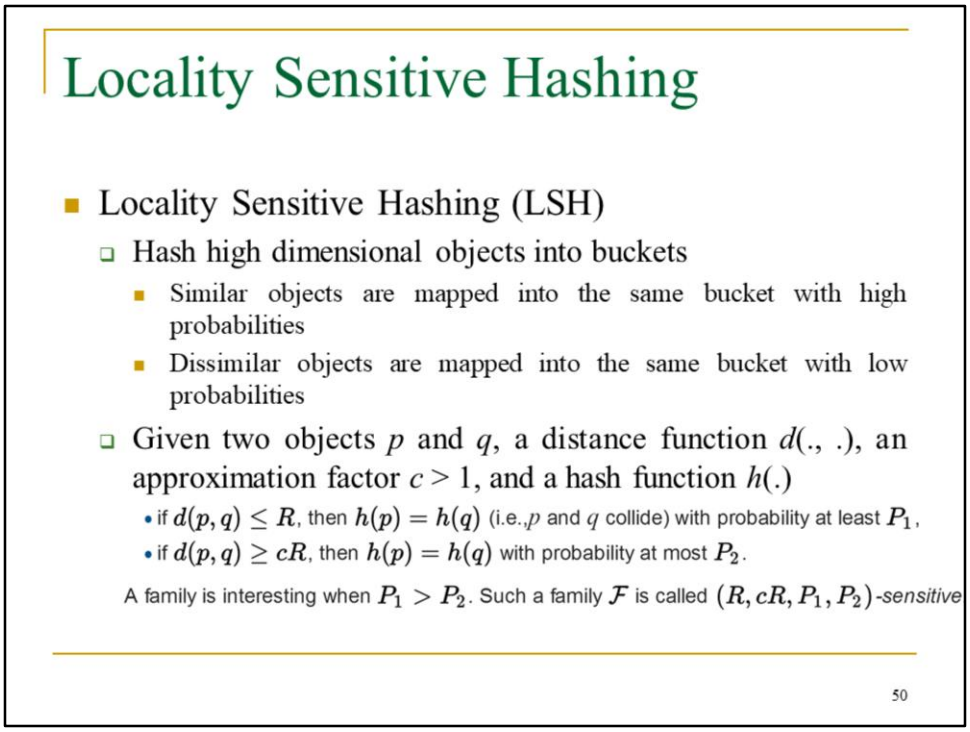
In our previous slide, we mentioned the trie index structure for text data. Here is an example showing a trie (or a prefix tree). We can search a word, by traversing the trie in the order of letters in the word. For example, if we want to search for word “to”, we can start from the root of the trie, select the left branch labeled by “t”, then left branch labeled by “o”, and finally access the word “to”.

Outline

- X-tree
- SS-tree, SR-tree
- M-tree
- Embedding-based index
- Inverted index
- Locality sensitive hashing

49

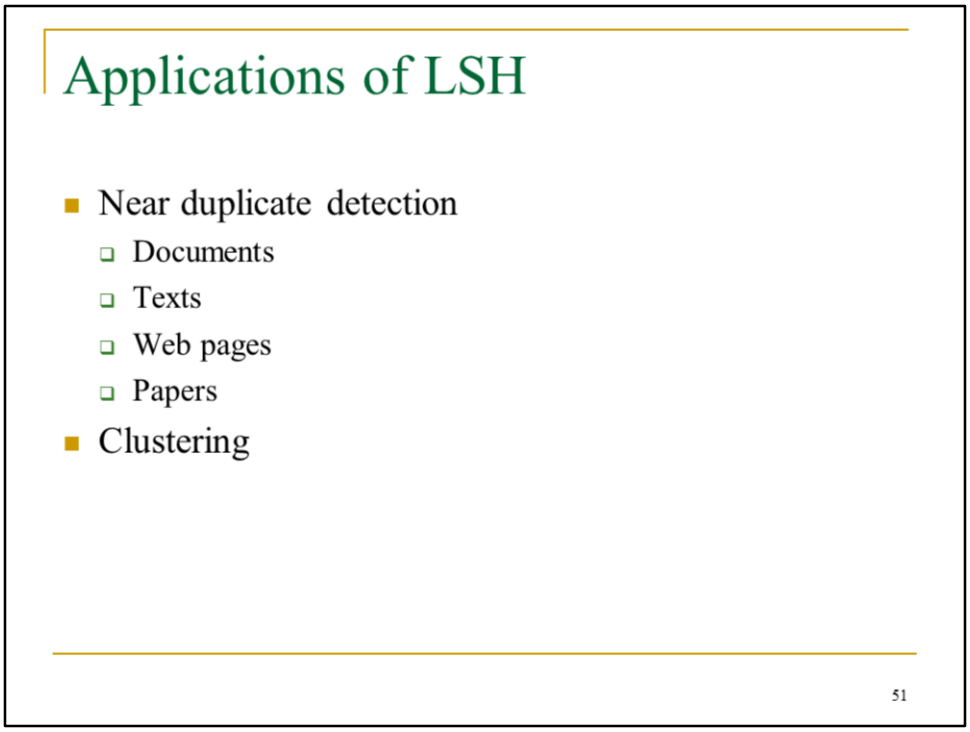
Finally, we discuss the locality sensitive hashing for spatial data.



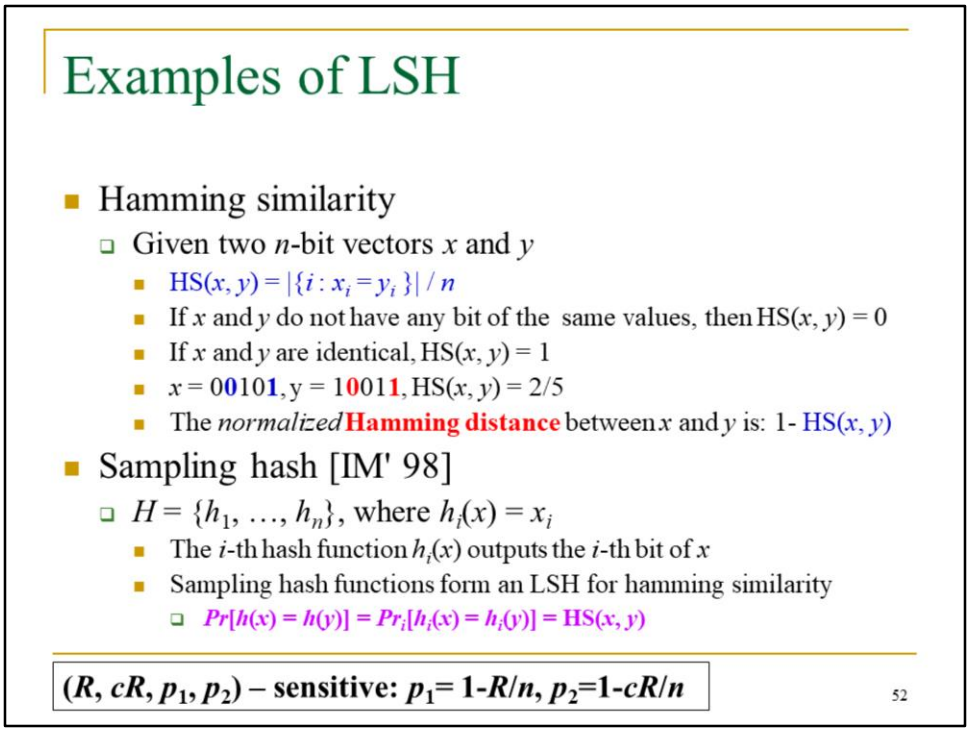
The locality sensitive hashing (LSH) is to hash high dimensional objects into buckets, such that similar objects are mapped to the same bucket with high probabilities, and dissimilar objects are mapped to the same bucket with low probabilities.

Formally, given two objects p and q, a distance function d(), an approximate factor c greater than 1, and a hash function h(), LSH satisfies 2 conditions that: (1) if the distance between p and q is less than a threshold R, then the probability that their hashed values are the same is at least P_1; (2) if the distance between p and q is greater than threshold cR, then the probability that their hashed values are the same is at most P_2. If the hashing function satisfies these two conditions, we call this hashing is a locality sensitive hashing.

Reference:
"Locality-Sensitive Hashing." *Wikipedia*, 19 Dec. 2018. *Wikipedia*, https://en.wikipedia.org/w/index.php?title=Locality-sensitive_hashing&oldid=874500999.



LSH is an approximate, but efficient, hashing method, which can be used for many real applications such as clustering, the near duplicate detection for documents, texts, Web pages, and papers, and so on.

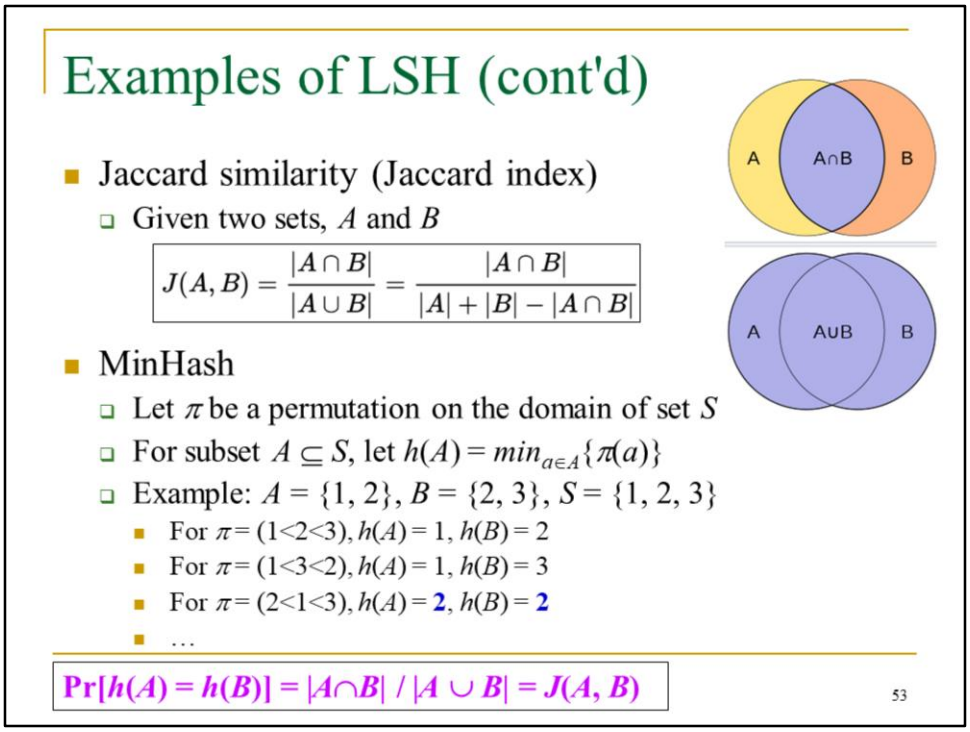


We will see some examples of LSH with different distance functions. First, we look into the hamming similarity between two bit vectors x and y. If x and y have different 0-1 values for all bits, then the hamming similarity between x and y is zero. If x and y are the same, then their hamming similarity is 1.

Hamming similarity is the percentage that two bit vectors have the same bit values. As shown in the example of the slide, two bit vectors have 2 positions with the same bit values, therefore, their hamming similarity is given by 2/5, where 5 is the size of the bit vector.

Next, we give the LSH in this scenario. Assume that we have n hashing functions h_i(x) = x_i, where n is the size of bit vector, and the i-th hashing function h_i(x) outputs the i-th bit of bit vector x, that is x_i. Then, the hamming similarity between x and y can be given by the probability that hashed values of x and y are the same by using randomly sampled hashing functions. We can prove that this is an LSH, where p_1 = 1-R/n and P_2 = 1-cR/n.

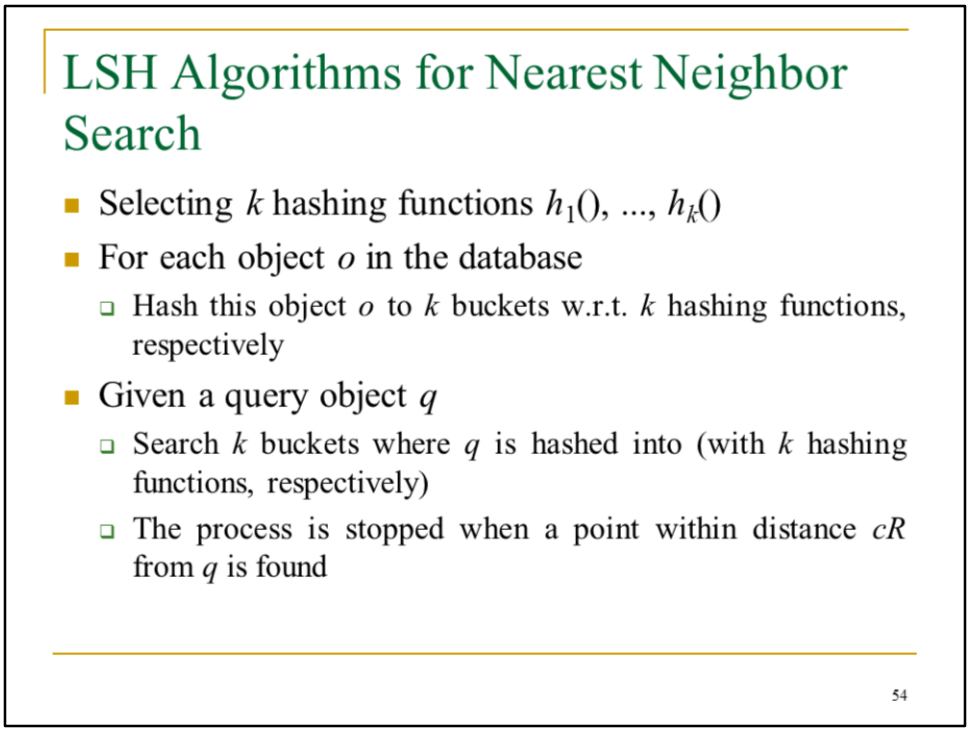
Reference:
Piotr Indyk and Rajeev Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In Proceedings of 30th Symposium on Theory of Computing, 1998.



Another LSH example is for Jaccard similarity between two sets A and B, which is given by the size of intersection between A and B divided by the size of A union B. We consider the MinHash. Let \pi be a permutation on the domain of set S. For any subset A of S, let the hashed value of subset A be the minimum value based on a permutation \pi of S.

As shown in the example, set A contains {1, 2}, set B contains {2, 3}, and domain S contains {1, 2, 3}. For different permutations \pi of S, we have different global orders among elements in S, for example, 1<2<3, 1< 3<2, and so on. Note that, here 1, 2, and 3 are elements in the set (which are not considered as numeric numbers). Therefore, with different permutations, we can obtain the hashed value of each subset. For example, for permutation 1<3<2, h(A) = 1, and h(B) = 3 (since 3<2 in the global order \pi). We can also prove that the Jaccard similarity is exactly the probability that the MinHash values from A and B are the same.

Reference:
"Jaccard Index." *Wikipedia*, 2 Dec. 2018. *Wikipedia*, https://en.wikipedia.org/w/index.php?title=Jaccard_index&oldid=871576429.



The LSH can be used for approximate nearest neighbor search. Specifically, we can define a family of LSH hashing functions. Each object in the database can be hashed into k buckets using k hashing functions, respectively. Then, given a query point for NN query, we can also hash q into k buckets and search these buckets. The approximate search stops when a point within cR-distance from q is found.

Reference:
"Locality-Sensitive Hashing." *Wikipedia*, 19 Dec. 2018. *Wikipedia*, https://en.wikipedia.org/w/index.php?title=Locality-sensitive_hashing&oldid=874500999.