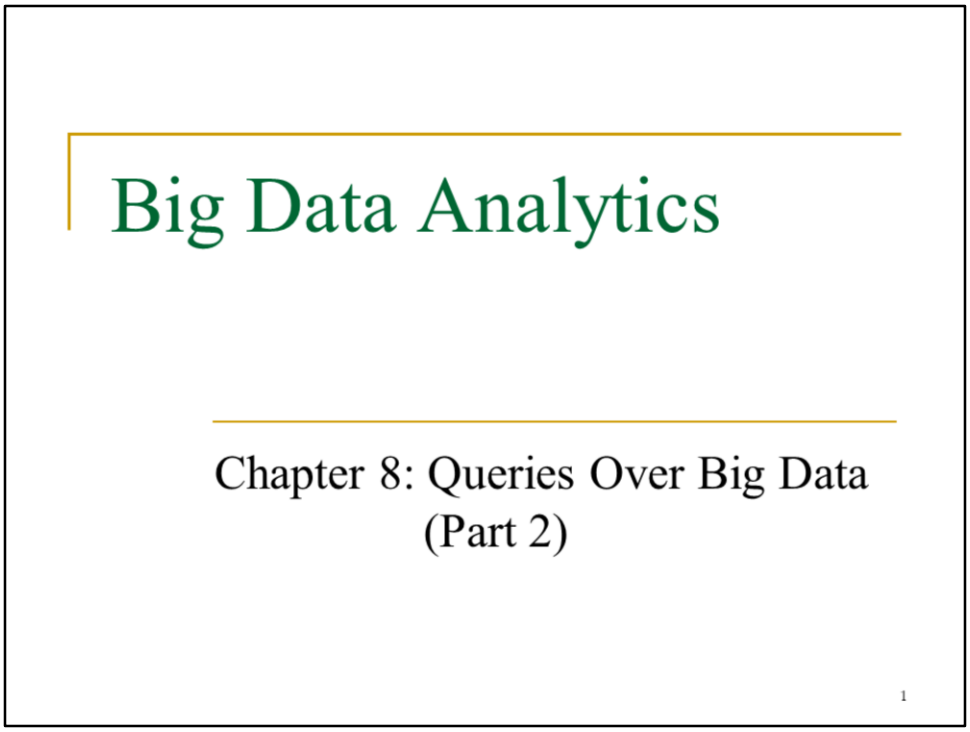
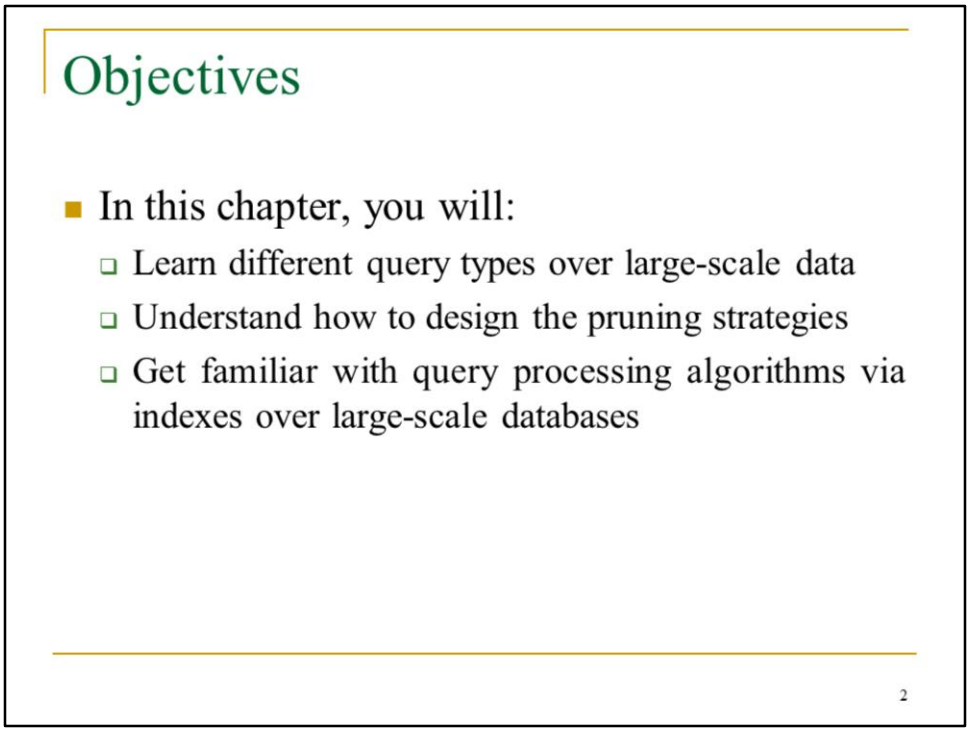
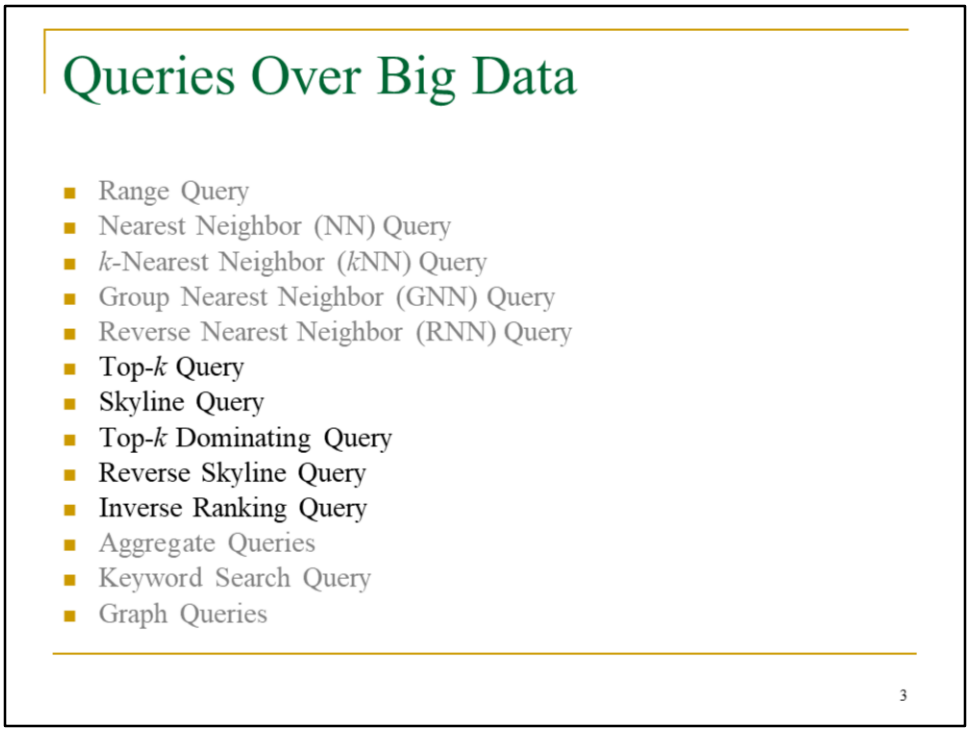


Chapter 8: Queries Over Big Data (Part 2)



In this chapter, we will continue to discuss more query types on big data.



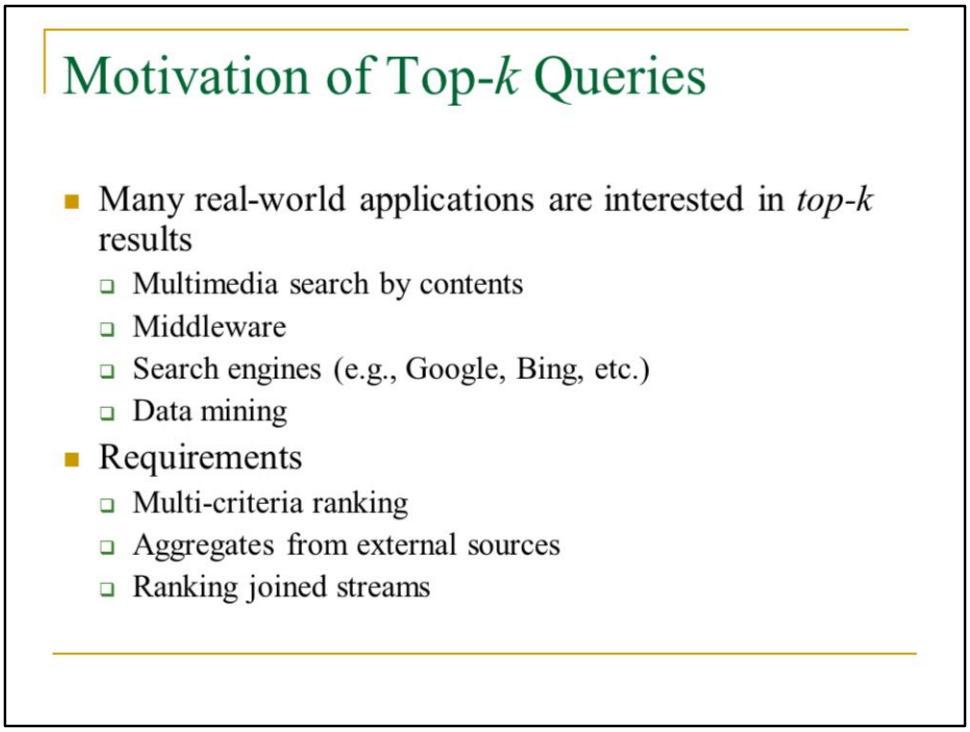
Specifically, we will consider the ranking-related queries, such as top-k query, skyline query, top-k dominating query, reverse skyline query, and inverse ranking query.

Outline

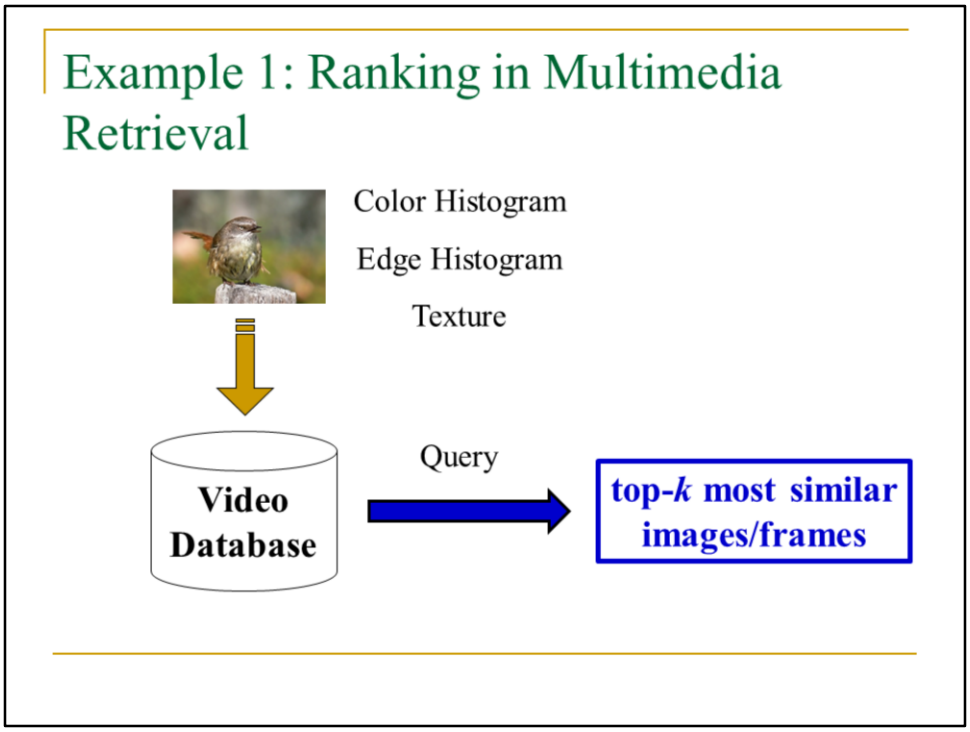
- Top-*k* Query
- Skyline Query
- Top-*k* Dominating Query
- Reverse Skyline Query
- Inverse Ranking Query

4

We will first start the discussion about the top-k query.



Let us first give the motivation for top-k queries, which rank and return k objects with the highest ranks. There are many real-world applications for top-k queries, such as multimedia search by contents, middleware, search engines, and data mining. In these applications, we usually need to rank objects with multiple criteria. Sometimes the ranking score needs the aggregates from external sources. Moreover, for some applications, we need to perform the ranking from multiple infinite streams.



References

Benjamint444. *White-Browed Scrubwren (Sericornis Frontalis, Cat.)English: A White-Browed Scrubwren Female (Sericornis Frontalis) Vocalising, Taken in South-Eastern Australia.* 1 Oct. 2007. Own work, *Wikimedia Commons*, https://commons.wikimedia.org/wiki/File:Scrub_wren_female_Vocalising444.jpg. Changes were not made.

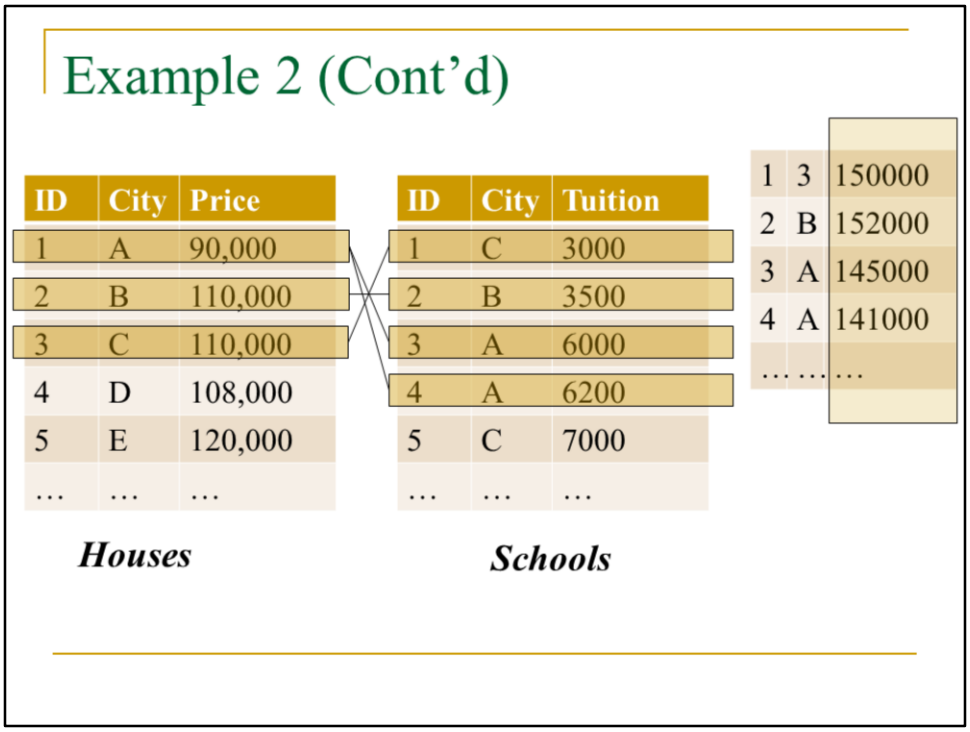
Examples of the top-k query include ranking in multimedia retrieval. From video database, we may want to retrieve k most similar frames to a given query image (or frame). Here, the similarity between two frames can be defined by a similarity function that involves color histogram, edge histogram, texture, and so on.

Example 2: SQL Over Relational Database

```
SELECT h.id , s.name
FROM houses h , schools s
WHERE h.location = s.location
ORDER BY h.price+10 x s.tuition
STOP AFTER 10
```

RANK( ) OVER in SQL 99

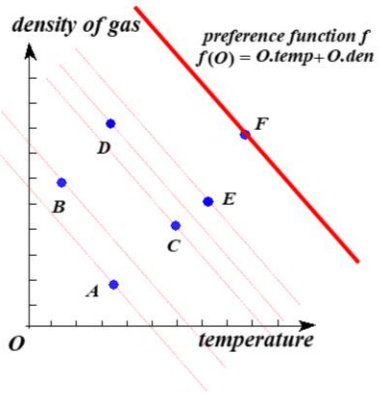
Another example of the top-k query is the SQL query in the traditional relational database, where we want to obtain 10 records about houses and schools at the same locations that have the highest house prices and tuition fees. In this example, the ranking function is given by directly summing up the house price and tuition fee, as shown in the SQL query of the slide.



Here is an example of house and school tables. From the previous SQL query, we need to join these two tables on "Location" attribute, then rank the joined results by the scores defined as the summation of "Price" and "Tuition", and finally return top-10 results with the highest scores.

Example 3: Coal Mine Surveillance

- In a coal mine surveillance application, a number of sensors are deployed to detect density of gas, temperature, and so on
- Assume we have a preference function $f(O) = O.temp + O.den$

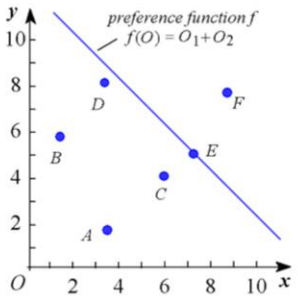


9

In the application of coal mine surveillance, we deploy sensors to tunnels in the coal mine, and collect sensory data such as the temperature, and the density of gas. Intuitively, high temperature and density of gas indicate high risks of dangerous events such as fire or explosion. In this example, if we simply define a preference function that sums up temperature and density of gas to rank each sensor, then the coal mine manager can issue a top-k query to monitor top-k sensor locations that have the highest risks of fire (or explosion).

Top-*k* Queries

- Given a spatial database D, a ranking function (or preference function) $f(o)$, and an integer parameter k, a *top-k query* retrieves k objects, o, from the database D such that they have the highest ranking score $f(o)$ among all objects in D



10

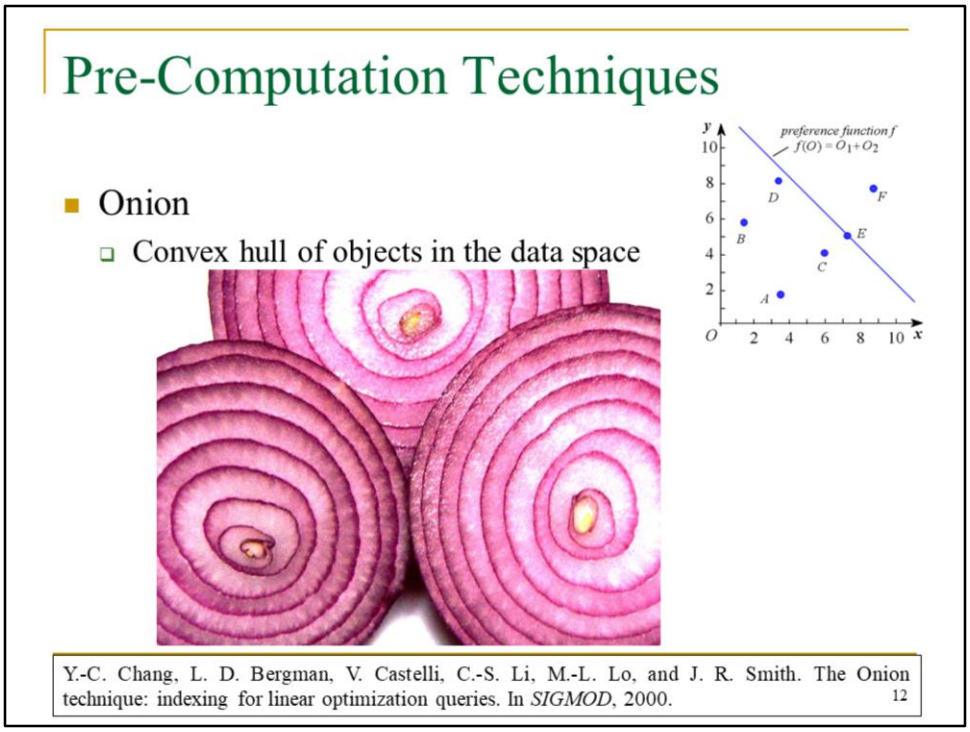
Formally, given a spatial database, a ranking function f(), and a parameter k, a top-k query returns k objects from the database, with the highest ranking scores f(o).

Ranking Functions

- Various ranking functions
  - $f(o) = o.x + o.y$
  - $f(o) = w_1 \cdot o.x + w_2 \cdot o.y$
  - ...

11

In practice, the user can specify different ranking functions such as the sum of attributes, weighted sum, and so on.



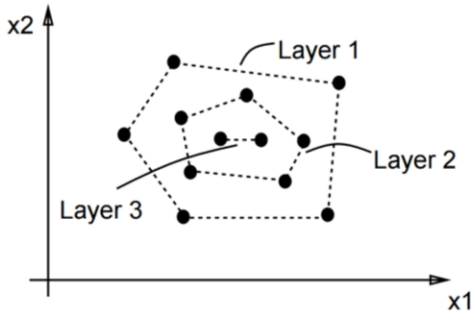
References

darwin Bell. Open Onion. 21 Nov. 2006. *Flickr*, https://www.flickr.com/photos/darwinbell/303892944/. Changes were not made.

To support online top-k query on objects, there are some existing works such as Onion and Prefer, which uses offline pre-computation techniques. In particular, the basic idea of the Onion approach is that top-k objects are usually located at the boundaries of the data sets. Therefore, it offline pre-computes the multi-level convex hulls for objects in the data space, which are candidates for top-k answers.

Onion for Top-*k* Queries

- Top-*k* answers must be in the first *k* layers of the convex hulls

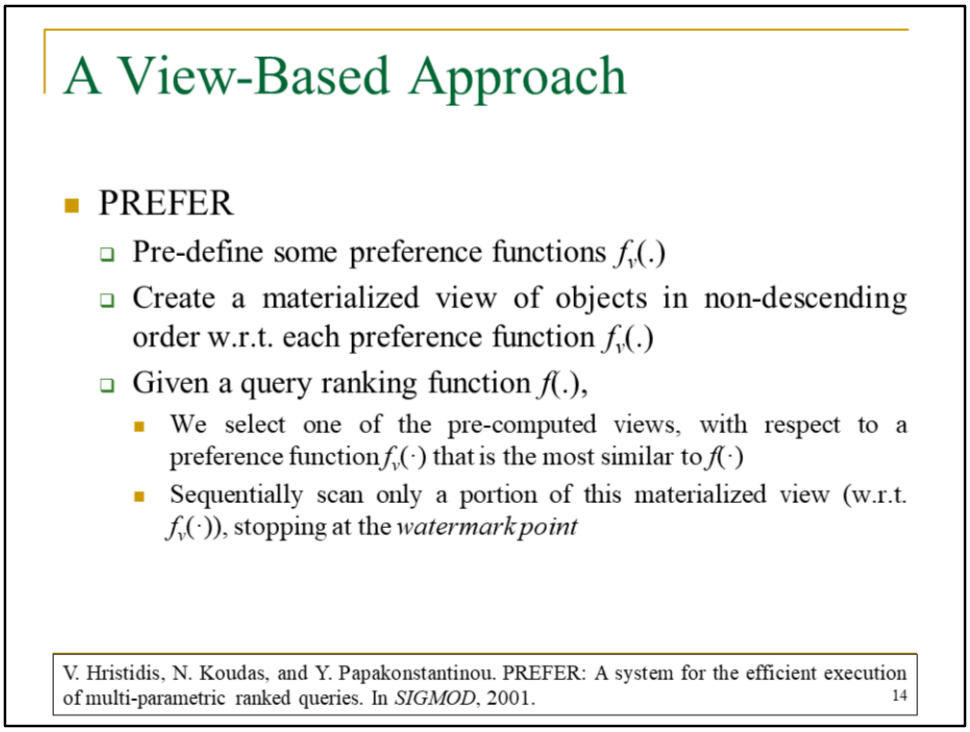


Y.-C. Chang, L. D. Bergman, V. Castelli, C.-S. Li, M.-L. Lo, and J. R. Smith. The Onion technique: indexing for linear optimization queries. In *SIGMOD*, 2000.

13

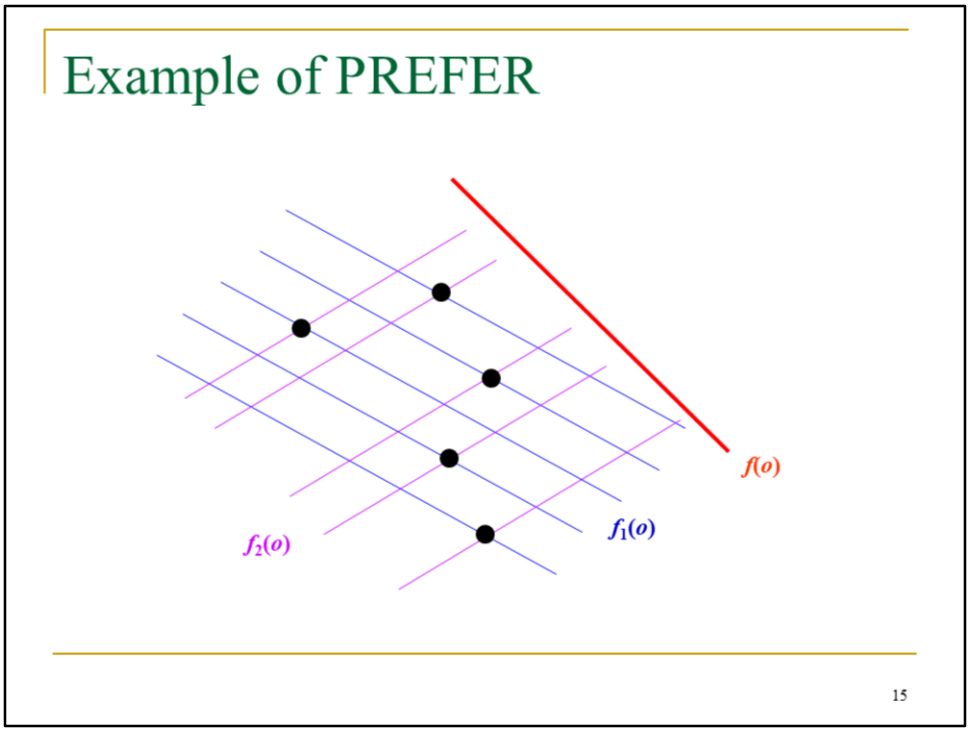
As shown in the figure, the Onion approach first computes the outermost convex hull of the data set, remove them from the data set, compute the convex hull of the remaining data objects as layer 2, and so on.

This way, if we want to retrieve top-k query answers, then the answers must be in the first k layers of convex hulls. For example, if k=1, then top-1 answer must be some object in Layer 1. Similarly, if k=2, then top-2 answers must be two objects from Layer 1 or 2.

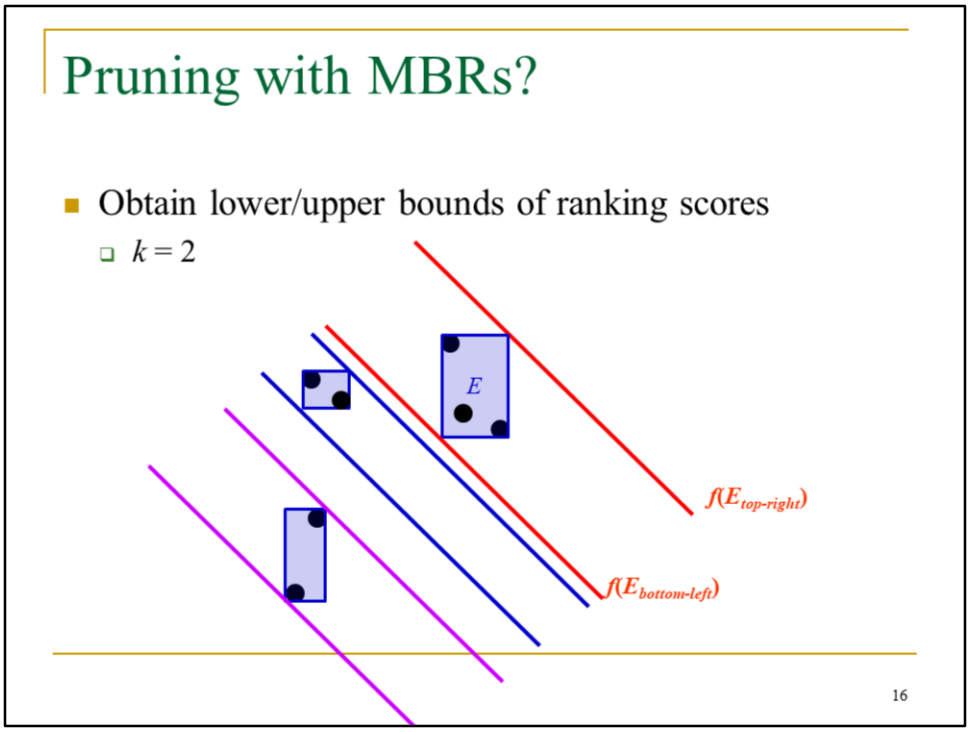


Another approach, PREFER, is a view-based method, which predefines some preference (or ranking) functions, and creates a materialized view of sorted objects for each preference function.

For any query ranking function, we can first find one pre-defined preference function that is most similar to the query preference function, and then use its corresponding materialized view of sorted objects to find top-k answers. Intuitively, if a pre-defined preference function is very similar to the query preference function, then we only need to access a very small portion of the materialized view, based on the pre-defined preference function.

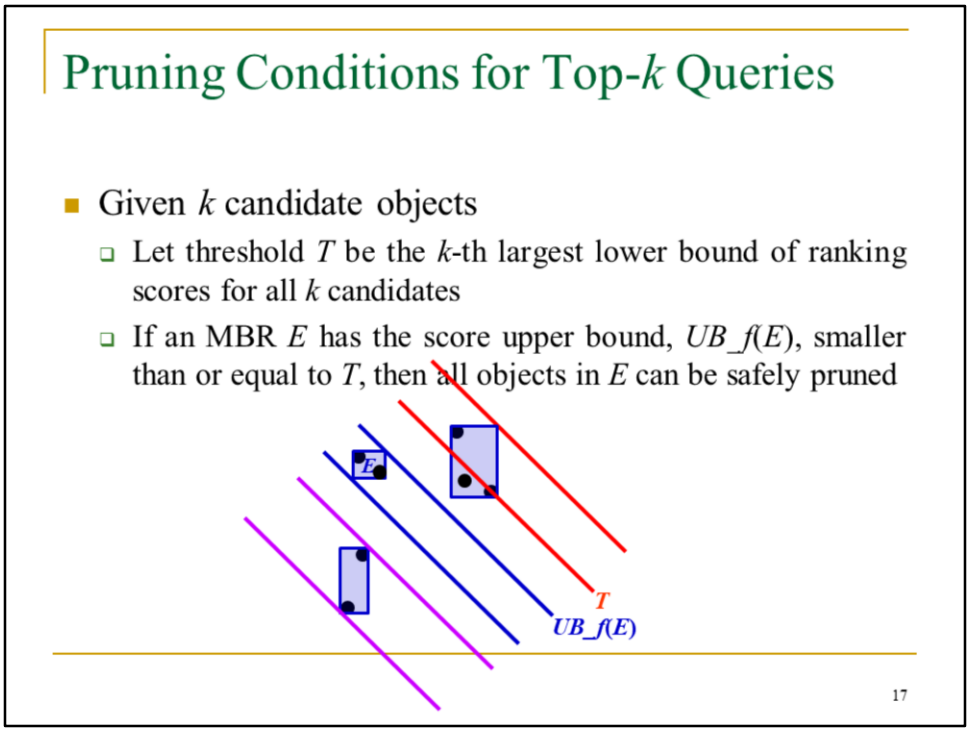


As an example in the figure, PREFER offline pre-computes the ranking of objects with respect to two preference functions, f_1() and f_2(). Then, for any given online query preference function f(), we first find one pre-computed ranking function that is the most similar to function f(). In this example, f_1() is more similar to f(). Next, we will access the offline sorted objects in descending order of f_1(), and obtain top-k objects o with the highest score f(o).



So far we have talked about two pre-computation techniques to obtain top-k answers. For online top-k query processing without offline pre-computations, we can also use the R-tree index to facilitate the query processing. For example, on the MBR node level, we can obtain the lower and upper bounds of scores for all objects in an MBR E. As shown in the figure, for preference function f() and MBR E, the score lower bound of MBR E is achieved at the bottom-left corner of the MBR E, whereas the score upper bound is achieved at the top-right corner. This way, we can obtain lower and upper bounds for all MBRs.

In the example of the figure, MBR E contains more than 2 (=k) objects, and its score lower bound is larger than the upper bound of all other MBRs. In this case, MBR node E contains top-k candidates, and we can safely prune other MBRs (since they cannot contain objects with high scores).



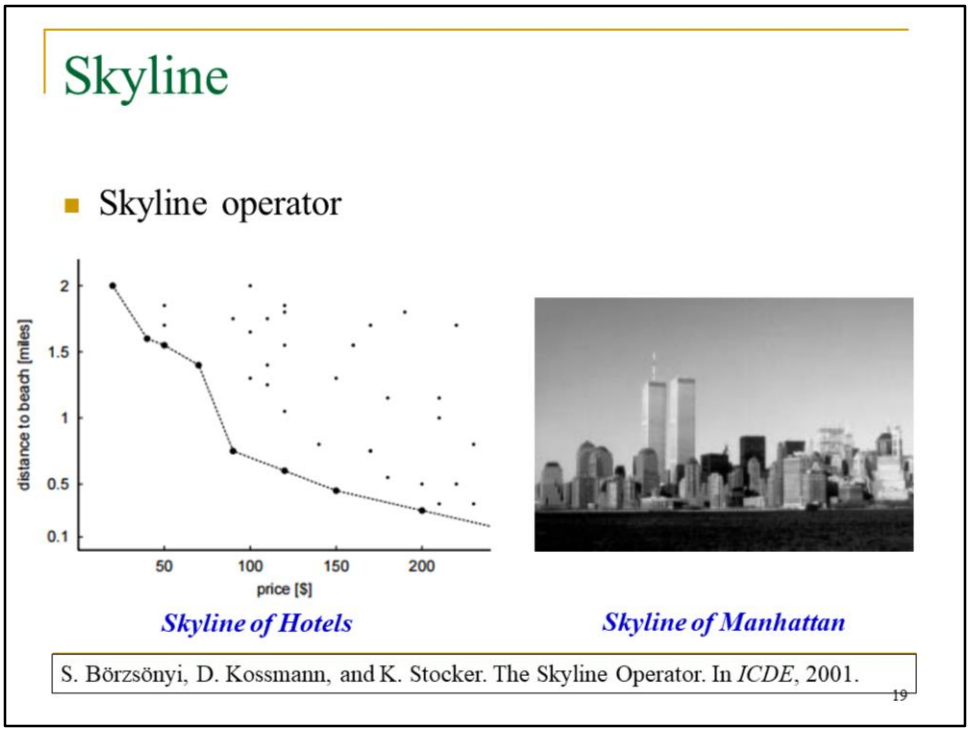
Similarly, if we have obtained k candidate objects, we can set a threshold T as the k-th largest lower bound of ranking scores. For any MBR E, if it has a score upper bound UB_f(E) smaller than or equal to T, then this MBR node E can be safely pruned. This is because, all objects in E must have scores lower than at least k candidates.

Outline

- Top-k Query
- Skyline Query
- Top-k Dominating Query
- Reverse Skyline Query
- Inverse Ranking Query

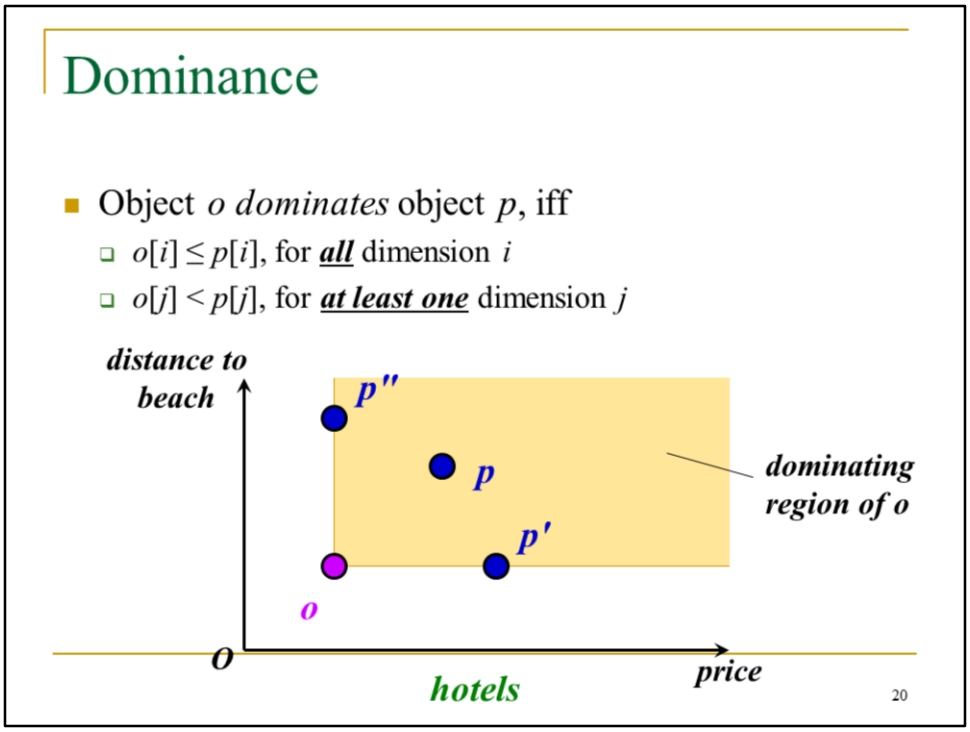
18

Next, we will talk about the skyline query.



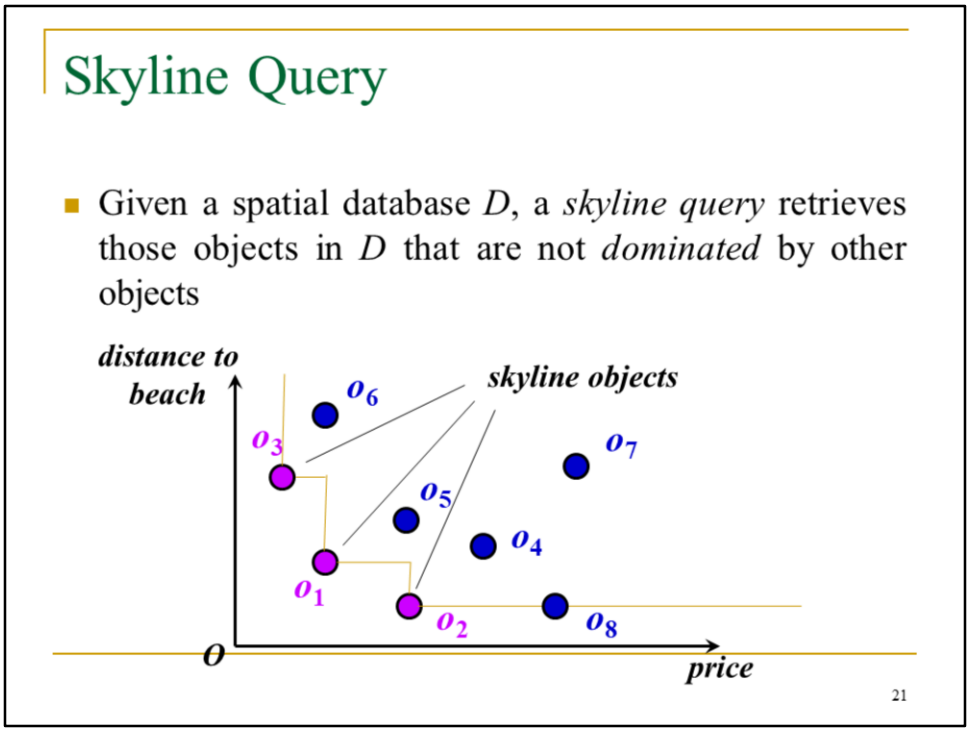
The first skyline paper was proposed in ICDE 2001. The right figure shows the skyline of Manhattan, and the skylines over objects in a 2D space are shown in the left figure.

The classic motivation example of the skyline query is the selection of hotels. Each hotel has two attributes, its price and distance to the beach. Intuitively, we want to select a hotel with low price and short distance to the beach. However, in practice, some hotel close to the beach might have higher price, and some hotel with low price has long distance to the beach. Usually there is a trade-off between these two factors/attributes. Therefore, the skyline query is going to return a number of hotels, such that there are no other hotels better than these returned hotels, in terms of both price and distance.

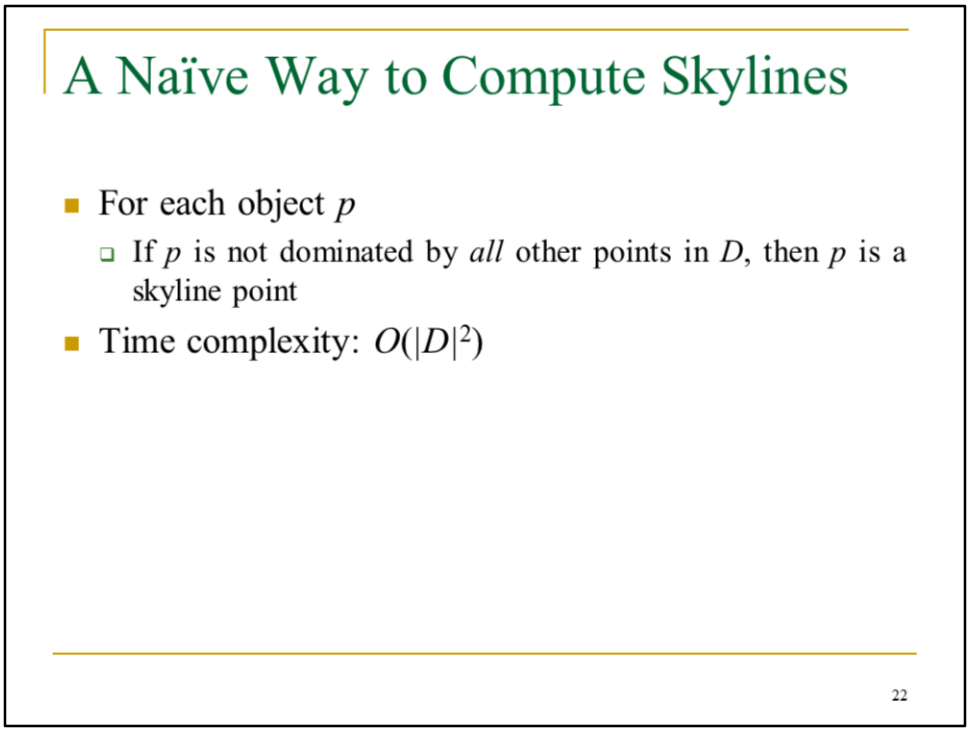


Formally, we first define the dominance relationship between two objects. In a multidimensional space, we say that object o dominates object p, if and only if two conditions hold: (1) for all dimensions, the attributes of object o are smaller than or equal to that of object p; and (2) for at least one dimension, object o has the attribute strictly smaller than that of object p.

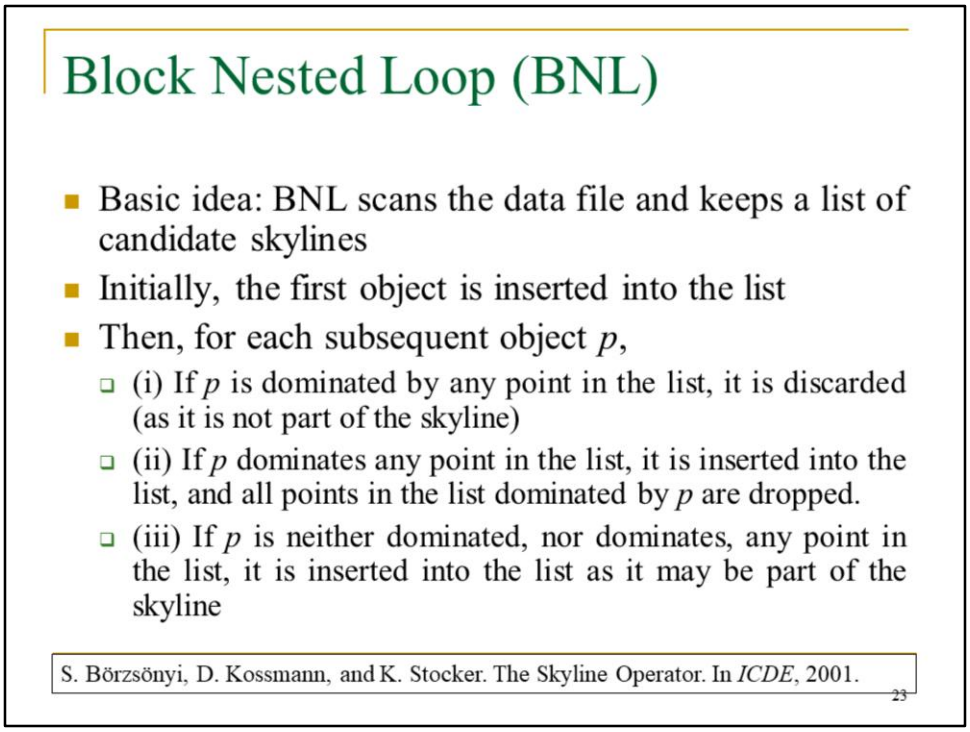
As shown in the figure, the shaded region is called the dominating region of hotel o. Any points inside the dominating region are dominated by hotel o (since they have both price and distance to beach worse than hotel o). In the example, we can see that, hotel o is dominating hotels p, p', and p".



For the skyline query, given a spatial database D, we want to retrieve those objects from database D such that they are not dominated by other objects. In the example of the figure, o_3, o_1, and o_2 are skyline points, since they are not dominated by other objects. For other objects, for example object o_6, which is dominated by object o_3, and thus it cannot be the skyline answer.



To answer the skyline query, one naïve way is to consider each object p, and compare the dominance relationship with all other objects in the database. If p cannot be dominated by all other objects, then p is a skyline point. This naïve method incurs quadratic cost with respect to the size of the database, which is not very efficient.

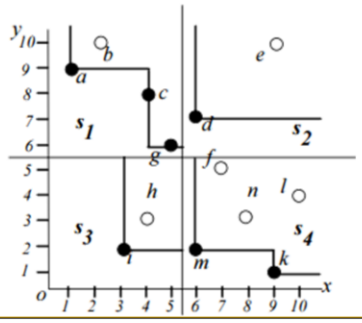


In the literature, there are many techniques proposed to answer skyline queries. When there is no index over the database, we can use the block nested loop (or BNL) method. The basic idea of BNL is as follows. Assume that all the data are stored sequentially on disk. BNL scans the entire data file once and keeps a list of candidate skylines during the scanning.

Initially, we insert the first object into the candidate list. Then, for every subsequent object p we encounter, we will compare it with candidates in the list. There are 3 cases. The first case is when p is dominated by some point in the list. In this case, we simply discard p (as p can never be the skyline). In the second case that p dominates some objects in the list, we insert object p into the list, and remove those dominated objects from the list. In the third case where p is neither dominated nor dominates any objects in the list, then p is also a possible candidate, and we also insert it into the list.

Divide-and-Conquer (D&C)

- D&C
  - Divides the data set into partitions, each fitting in memory
  - Compute partial skylines for each partition
  - Merge partial skylines into a final skyline set

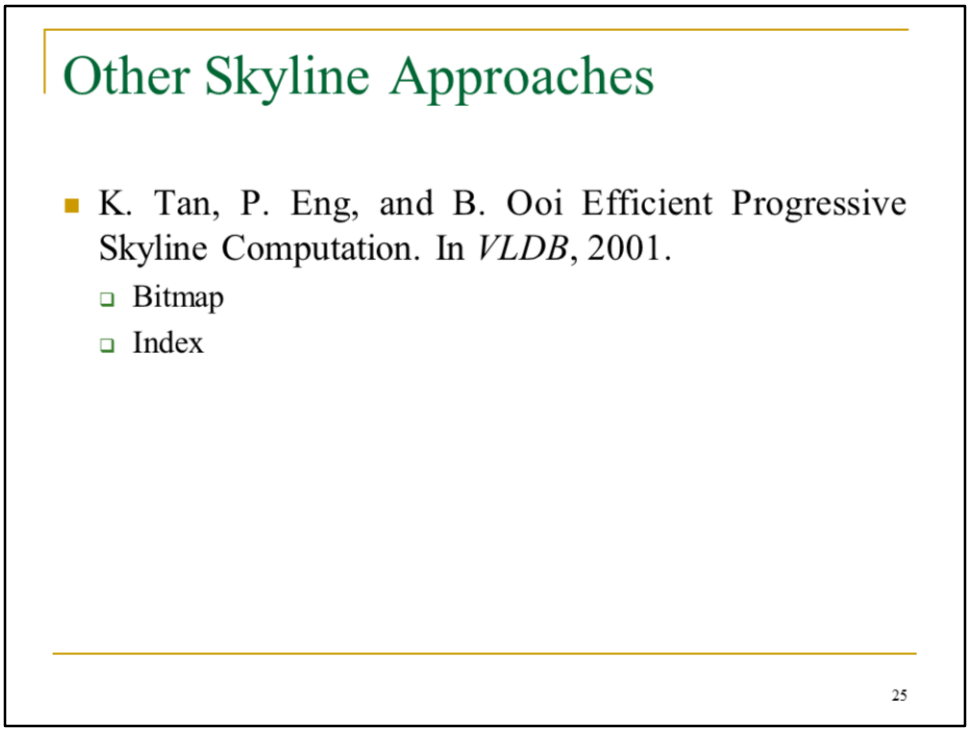


S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *ICDE*, 2001.

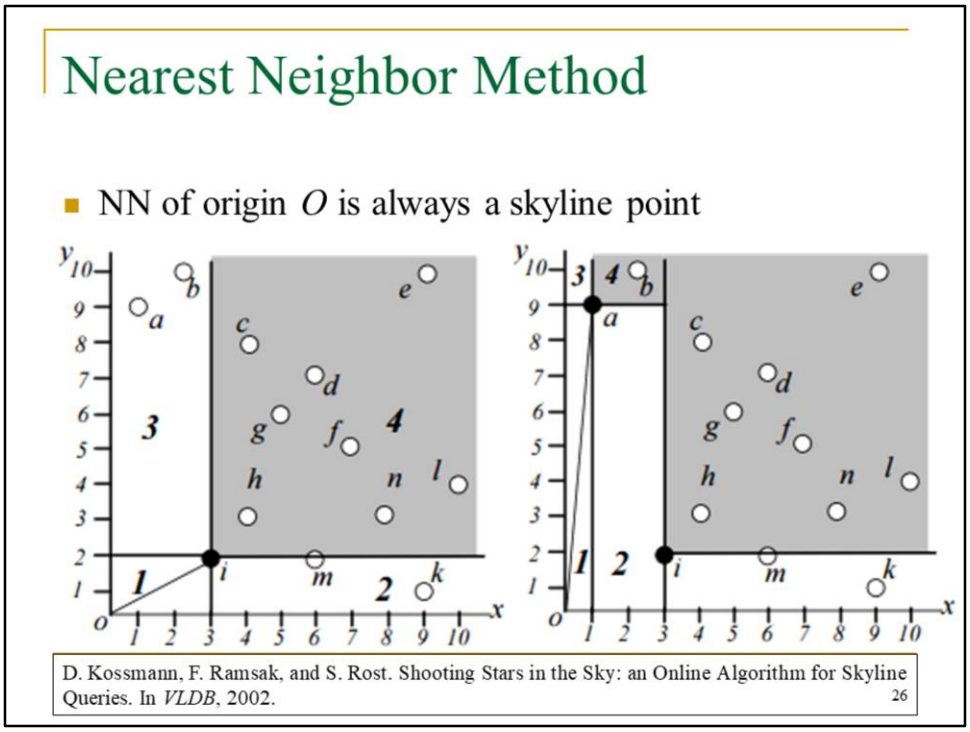
24

Another skyline computation approach without index is called divide-and-conquer (D&C) approach. This approach divides the data set into partitions, such that each partition can fit in memory. Then, we compute potential partial skylines within each partition, and merge them to obtain a global skyline set.

As shown in the figure, we divide the data set into 4 partitions s_1 ~ s_4, and compute skylines for each partition. For example, for partition s_1, objects a, c, and g are partial skylines in s_1. At the end, we combine partial skylines from 4 partitions and calculate the final skyline answer set {a, l, k}.



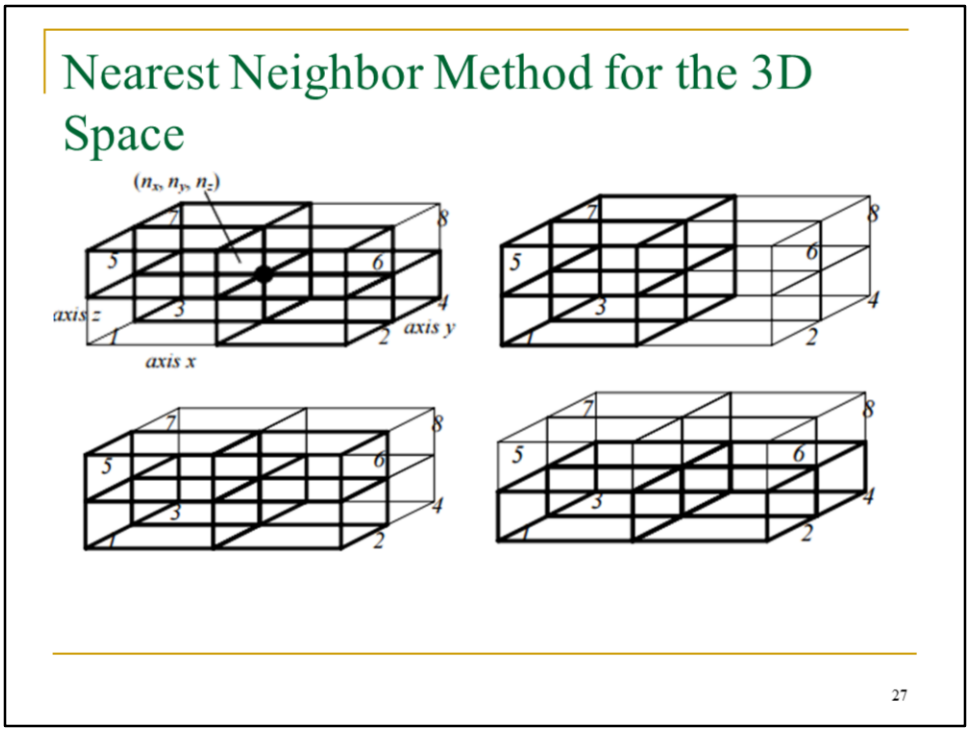
There are some other skyline processing approaches such as bitmap and index approaches. If you are interested in finding out more about these approaches, please read the VLDB 2001 paper.



There is another VLDB 2002 paper about the nearest neighbor approach to solve the skyline problem. The nearest neighbor approach is based on an observation that the nearest neighbor of the origin O is always a skyline point. Therefore, the basic idea of the nearest neighbor method is to first compute the nearest neighbor of the origin. As shown in the left figure, object i is the NN of origin O. Thus, object i is one of the skyline answers.

Next, with respect to the NN of O, we divide the 2D data space into 4 partitions, as shown in the left figure. The top-right partition cannot contain any skylines, since they are all dominated by object i. The bottom-left partition cannot contain any skyline either, since object i is the nearest neighbor of O. No other objects should fall into bottom-left partition.

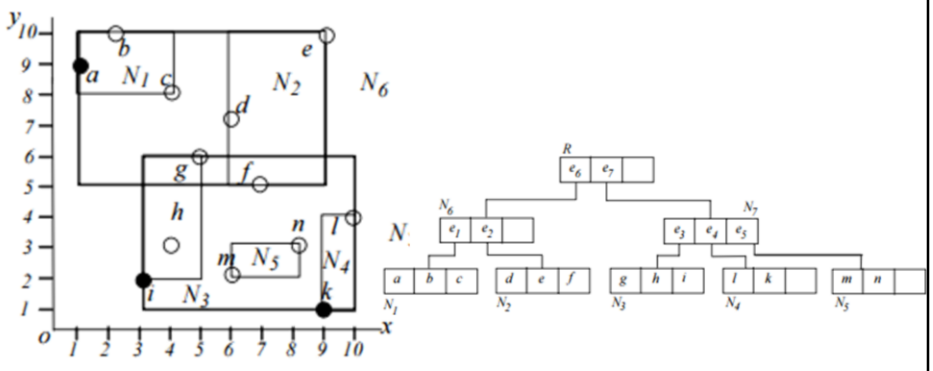
We thus only need to check the remaining 2 partitions. As shown in the right figure, we recursively apply the NN method by finding the nearest neighbor of origin O in each partition. For top-left partition, we can obtain NN object a. Similarly, for bottom-right partition, we can retrieve NN object k. This way, we can recursively find NNs in partitions, and obtain all skylines a, i, and k.



In higher dimensional space like 3D space, we can obtain 2^3 = 8 partitions with respect to an NN. To save the NN computation cost, we can group partitions by considering left 4 partitions, back 4 partitions, and bottom 4 partitions, as shown in the figures.

Branch and Bound Skyline (BBS)

- BBS uses R-tree to retrieve the skyline objects

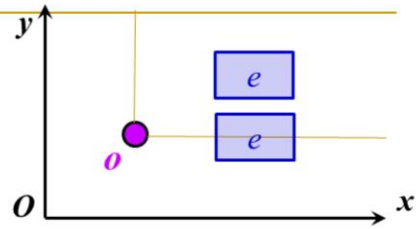


D. Papadias, Y. Tao, G. Fu, and B. Seeger. An Optimal and Progressive Algorithm for Skyline Queries. In *SIGMOD*, 2003.

28

Another index-based skyline approach is the branch and bound skyline (or BBS), which uses R-tree to facilitate the skyline query processing.

BBS Algorithm



Algorithm BBS (R-tree R)
1. $S=\emptyset$ // *list of skyline points*
2. insert all entries of the root R in the heap
3. while heap not empty
4. remove top entry e
5. if e is dominated by some point in S discard e
6. else // *e is not dominated*
7. if e is an intermediate entry
8. for each child e_i of e
9. if e_i is not dominated by some point in S
10. insert e_i into heap
11. else // *e is a data point*
12. insert e_i into S
13. end while

End BNN

BBS is proved to be I/O optimal!

29

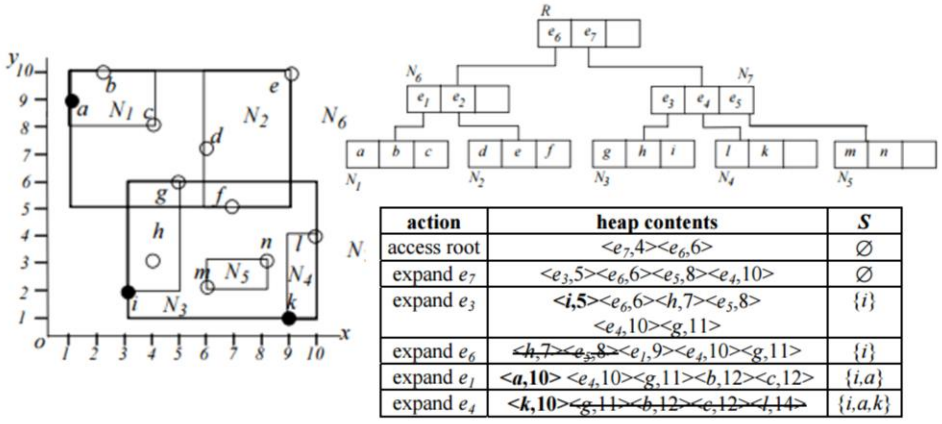
Here is the pseudo code of the BBS algorithm. We start from the root of the R-tree and initially insert entries of the root into heap. Each time we pop out an entry from the heap. Here, the order to access node entries in the heap is in ascending order of mindist() from origin to the node MBR. If the entry e is dominated by some point in a skyline candidate set S, then we can safely discard this entry e (as no points under e can be skylines). Otherwise, we need to access children of this entry e.

If entry e is a non-leaf or leaf node, then we check if any of its children is dominated by some point in S. If the answer is yes, then we can prune that child entry; otherwise we insert it into the heap for further checking.

If entry e is a data point, then we simply add it to candidate set S (as it has not been dominated by other objects seen so far).

The BBS algorithm is proved to be I/O optimal, in the sense that those MBR nodes that do not contain skylines will not be accessed.

Example of BBS Algorithm



action	heap contents	*S*
access root	<e_7,4><e_6,6>	∅
expand e_7	<e_3,5><e_6,6><e_5,8><e_4,10>	∅
expand e_3	**<*i*,5>**<e_6,6><*h*,7><e_5,8> <e_4,10><*g*,11>	{*i*}
expand e_6	~~<*h*,7><e_5,8>~~<e_1,9><e_4,10><*g*,11>	{*i*}
expand e_1	**<*a*,10>** <e_4,10><*g*,11><*b*,12><*c*,12>	{*i*,*a*}
expand e_4	**<*k*,10>**~~<*g*,11><*b*,12><*e*,12><*l*,14>~~	{*i*,*a*,*k*}

30

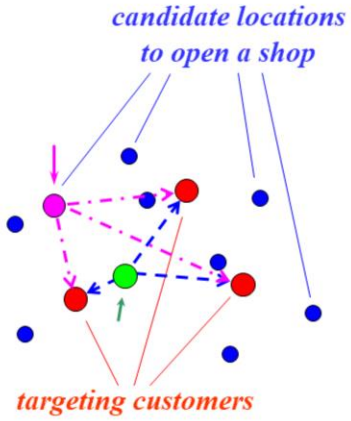
Here is an example of the BBS algorithm. Initially we add 2 entries of the root, e_6 and e_7, to the heap. Then, we expand top entry e_7 from the heap, and insert entries e_3, e_4, and e_5 into heap. Next, we expand node e_3, and obtain objects g, h, and i. Intuitively, object i is a NN of the origin, which is a skyline added to the candidate list S.

For the next step, we expand node e_6, insert e_1 into the heap, and discard e_2 (as it is dominated by candidate i in S). Then, we delete object h and node e_5 (which are dominated by candidate i in S), and expand node e_1. Next, we insert a, b, and c into the heap. Since object a is not dominated by i, it is added to the candidate list S. Finally, we expand node e_4, insert object k into candidate list S (as it is not dominated by objects i and a), and discard object l (as it is dominated by object i). The remaining objects in the heap are dominated by objects in S, and can thus be pruned.

Therefore, objects i, a, and k in the candidate list S are skylines and returned.

Motivation Example of Spatial Skylines

- Assume we have a number of candidate locations to open a shop in a city
- The shop is targeting at customers in some residential areas
- The problem is to find out appropriate locations such that there are no better locations in terms of *travelling distances* to customers

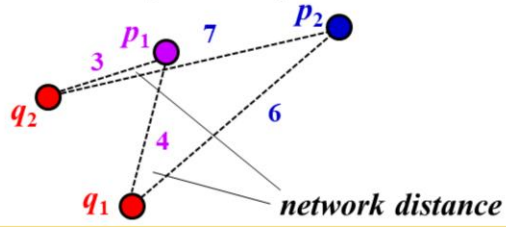


31

There are many variants of the skyline query since 2001. One of them is called the spatial skyline query. Assume that there are many candidate locations to open a shop in a city. The shop is targeting at customers in some residential areas. In this example, residential areas are query points and candidate shop locations are objects.
The spatial skyline problem is to return appropriate candidate shop locations such that their traveling distances from residential areas are better than (or not dominated by) others.

Spatial Skyline or Multi-Source Skyline Query

- Attributes of objects are ***not static***, but ***dynamically computed*** w.r.t. a number of query points
- Given a set, Q, of n query points q_i, each object o_j has the dynamic attribute $dist_N(o_j, q_i)$ on the i-th dimension
- A multi-source skyline query retrieves those objects that are not dynamically dominated by other objects



M. Sharifzadeh and C. Shahabi. The Spatial Skyline Queries. In *VLDB*, 2006.

K. Deng, X. Zhou, and H. T. Shen. Multi-source skyline query processing in road networks. In *ICDE*, 2007.

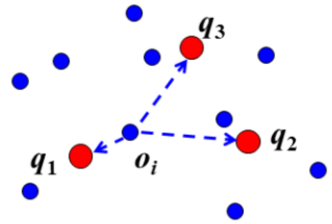
32

Formally, each object in the database has n dynamic attributes, which are defined as the distances to n query points, respectively. Then, with dynamic attributes of objects, the spatial skyline (or called multi-source skyline) query obtains those objects whose dynamic attributes are not dominated by others.

Here, the distance function can be the road-network distance (i.e., the shortest path distance) in the graph.

The Metric Skyline Query

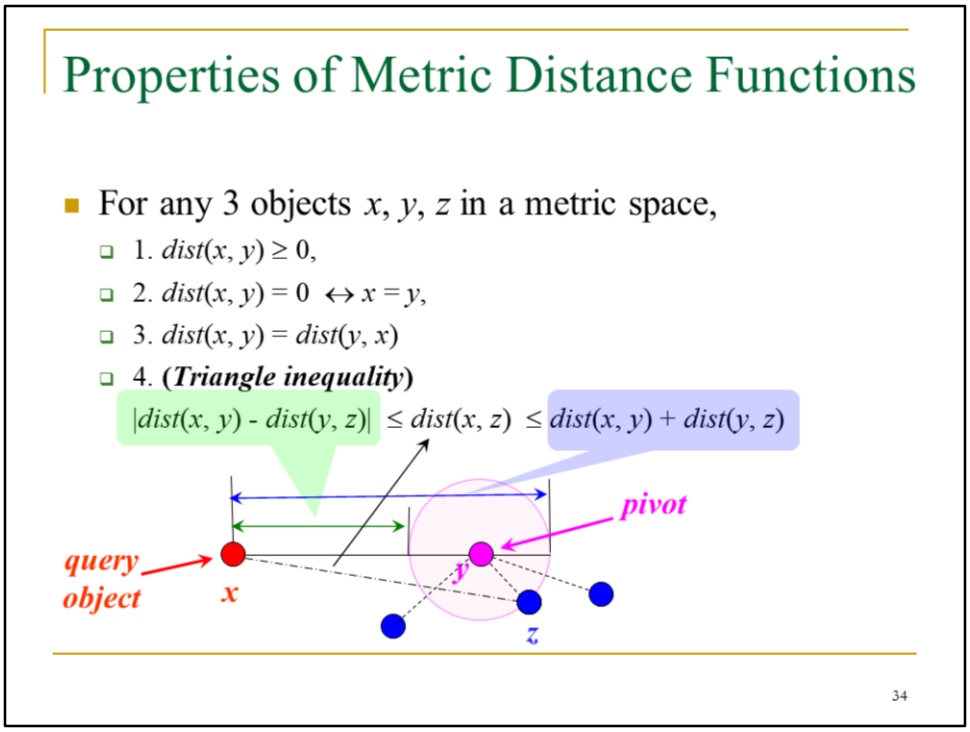
- Metric Skyline
  - Given a database D with m data objects in a *metric* space, and a set of query objects $Q = \{q_1, q_2, \ldots, q_n\}$,
    - A *metric skyline* query retrieves all the objects o_i such that their dynamic attribute vectors $<dist(o_i, q_1), dist(o_i, q_2), \ldots, dist(o_i, q_n)>$ are not *dominated* by those of other objects, where $dist(., .)$ is a *metric* distance function



L. Chen and X. Lian. Dynamic skyline queries in metric spaces. In *EDBT*, 2008.

33

In a general case, we can consider the multi-source skyline problem in the metric space, called the metric skyline query. Note that, the network distance is only a special case of metric distance. In the metric space, the metric skyline query obtains those objects that are not dynamically dominated by others, where dynamic attributes of objects are defined as the metric distance between two objects in the metric space.

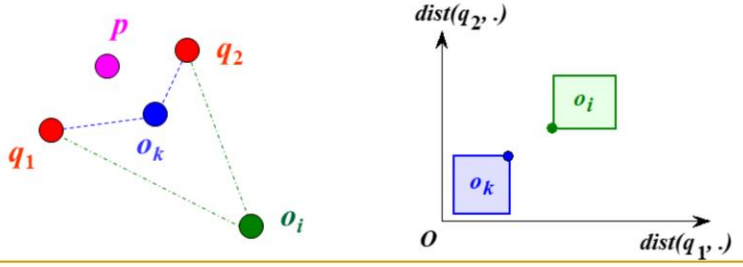


Specifically, the metric distance function satisfies non-negativity, that is, the distance between two objects is always non-negative, and zero is achieved for the distance between the same objects (called identity property). It also satisfies the symmetry property, that is, the distance from x to y is the same as that from y to x.

The last property is the triangle inequality, which is the most important property of the metric distance function. The basic idea of the triangle inequality is that the distance from x to y plus that from y to z is always greater than or equal to the distance from x to z. With this property, we can derive the lower and upper bounds of the distance from a query point x to any object z, via a pivot y. For online queries, we do not have to compute the distance between query point x and all other objects (like z). We can offline pre-compute the distances from pivot y to all objects (like z). Then, for online queries, we only need to compute the distance from query point x to pivot y once, and obtain lower/upper distance bounds from x to z via pivot y, by using the triangle inequality.

Triangle-Based Pruning

- Denote $LB_{ij} = |dist(q_j, p) - dist(p, o_i)|$ and $UB_{ij} = dist(q_j, p) + dist(p, o_i)$, where p is a *pivot*
- Given a database D, and a set of query points $Q = \{q_1, q_2, \ldots, q_n\}$, a data object o_i can be safely pruned if there exists an object o_k such that $UB_{kj} \leq LB_{ij}$ for *all* $1 \leq j \leq n$

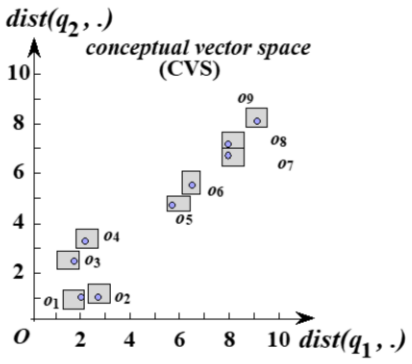


35

Based on the property of the triangle inequality in the metric space, we can derive lower and upper bounds of the distances from each object to query points in the spatial skyline problem. In the example of the figure, we have two query points q_1 and q_2, and two objects o_i and o_k, as well as a pivot p. With the triangle inequality, we can obtain distance bounds from each object to the two query points, and transform each object to a converted 2D space, where each attribute of the 2D space is the distance from object to a query point (e.g., q_1 or q_2).

Conceptual Vector Space (CVS)

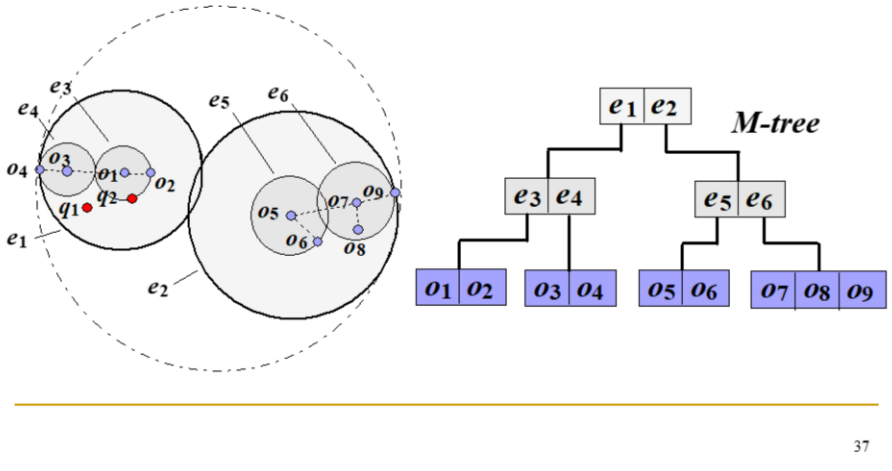
- By the triangle inequality, we can convert each data object o_i into a hyperrectangle in an attribute (vector) space, namely *conceptual vector space* (CVS), where each dimension corresponds to $[LB_{ij}, UB_{ij}]$
- Intuitively, we want to conduct a skyline computation in CVS



We call the converted space conceptual vector space (CVS), in which each object is represented by an MBR. For the metric skyline problem, we can conduct the skyline computation in this converted space.

Indexing

- We assume data are indexed by M-tree



37

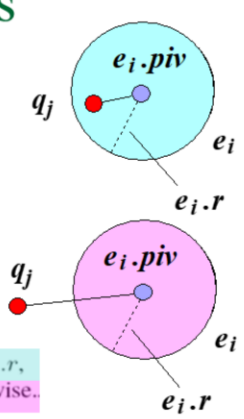
Since all objects are in the metric space, we can use a metric index, M-tree, as mentioned earlier to index objects.

Pruning Intermediate Nodes

- Let e_i be an intermediate node of M-tree with pivot $e_i.piv$ and radius $e_i.r$
- By the triangle inequality, we have:
  - For any object $o_x \in e_i$
  - $dist(q_j, o_x) \in$

$$\begin{cases} [0, dist(q_j, e_i.piv) + e_i.r] & \text{if } dist(q_j, e_i.piv) \leq e_i.r, \\ [dist(q_j, e_i.piv) - e_i.r, dist(q_j, e_i.piv) + e_i.r] & \text{otherwise.} \end{cases}$$

  - A node e_i can be safely pruned if lower bound of $dist(q_j, o_x)$ is greater than UB_{kj} for *all* $1 \leq j \leq n$, for some candidate o_k



38

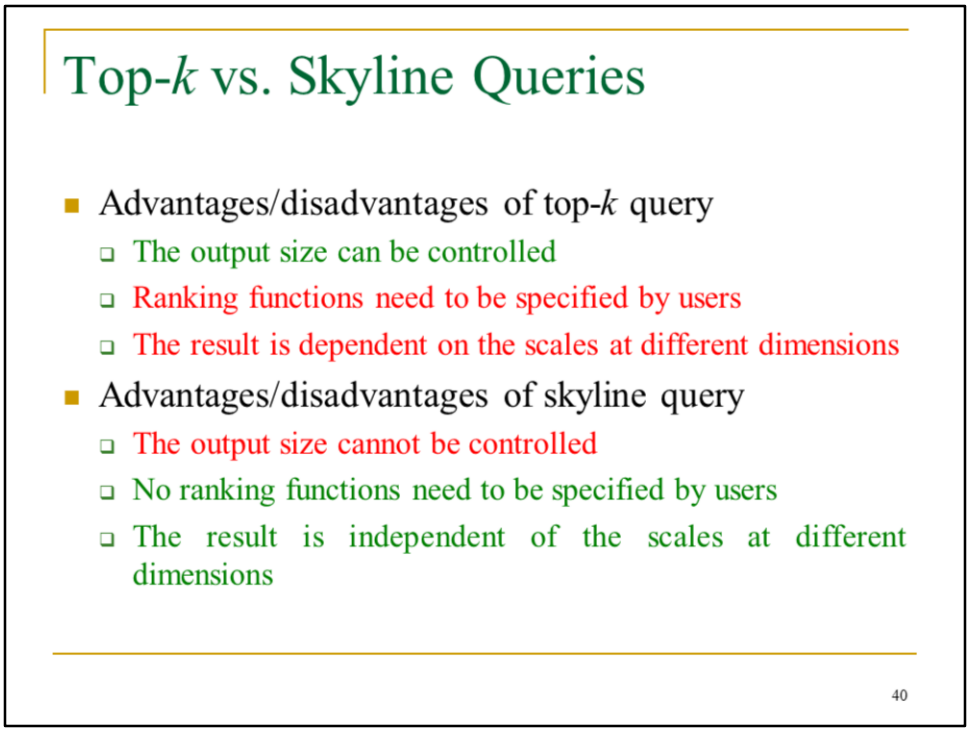
For intermediate node e_i of the M-tree, we can also derive the lower/upper bounds of distances from a query point q_j to any object o_x inside e_i, as shown in the formulae. Any node e_i can be safely pruned, if it holds that the lower bound of dist(q_j, o_x) is greater than the distance upper bound w.r.t. some candidate object o_k, for all dimensions (in other words, all objects o_x in e_i are dominated by o_k in the converted space).

Metric Skyline Query Processing

- We index data objects in the metric space with an M-tree
- We traverse the M-tree in a *best-first* manner such that skylines are implicitly computed in CVS
  - Maintain a minimum heap with entries in the form (e, key), where e is a point/node, and key is a summation of minimum distances from e to q_j, for $1 \leq j \leq n$

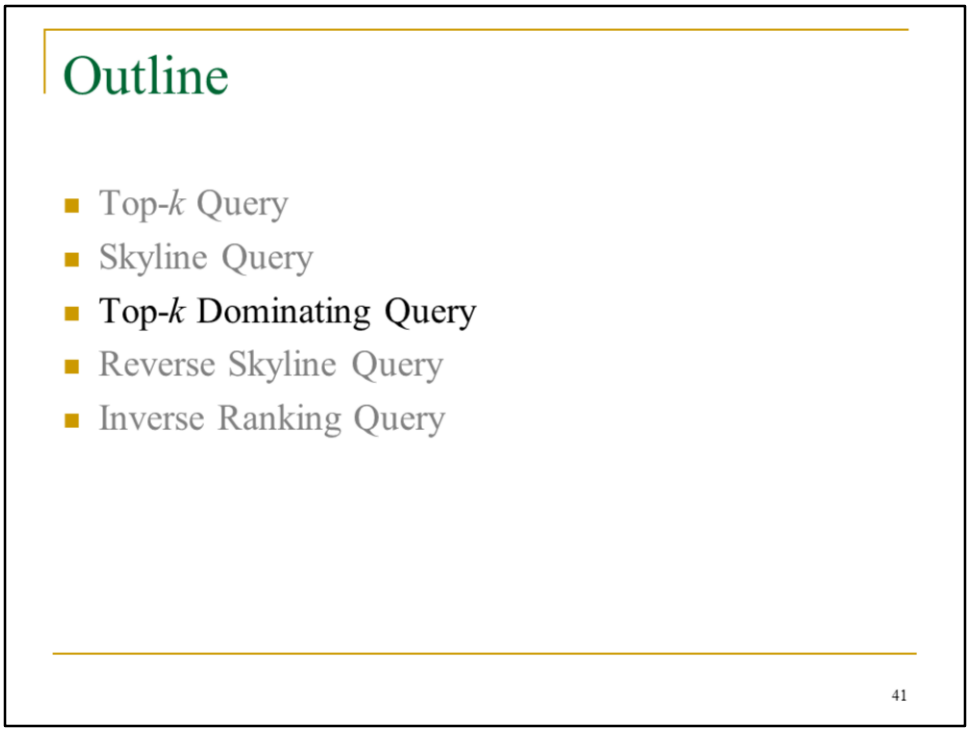
39

To answer the metric skyline query, we can traverse the M-tree in a best-first manner such that the metric skylines are implicitly computed in the converted space. Specifically, we can use a minimum heap to traverse the M-tree index, similar to R-tree traversal. For details, please refer to the metric skyline paper.



So far we have talked about the top-k and skyline queries. Each of them has its own advantages and disadvantages. For the top-k query, we can control the parameter k such that we can get exactly k query answers. However, the user needs to have the domain knowledge in order to specify the ranking function, which is not trivial. For example, it is not trivial how to define a ranking function between temperature and humidity, and what balancing parameter between these two attributes. Moreover, the top-k results highly depend on the scales of the attributes. If we change the scale of even one attribute (e.g., multiplying 100), then top-k results may be totally different. Thus, top-k query answers are very sensitive to the scales of attributes.

On the other hand, for the skyline query, the size of the skyline answer set cannot be controlled. For very high dimensional space, almost any object in the data set is a skyline. Nevertheless, users do not need to specify any ranking functions for issuing the skyline query, and the skyline results are independent of the scales at different dimensions.



Inspired by the advantages and disadvantages of top-k and skyline queries, the top-k dominating query was proposed, which does not have the disadvantages of top-k or skyline query.

Top-*k* Dominating Query

- Skyline in high dimensional spaces
  - Almost all objects are skylines
  - Meaningless!
- Solution: Rank objects by scores!
  - How to define the score function $f(o)$?
  - $f(o)$ is the number of objects *dominated* by object o

- The output size can be controlled
- No ranking functions need to be specified by users
- The result is independent of the scales at different dimensions

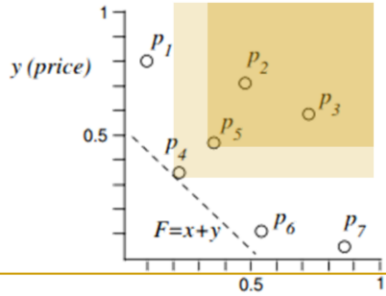
M. L. Yiu and N. Mamoulis. Efficient Processing of Top-*k* Dominating Queries on Multi-Dimensional Data. In *VLDB*, 2007.

42

As mentioned in the previous slide, the skyline query would return almost all objects in the database as skylines in high dimensional space, which are not meaningful. The top-k dominating query can control the returned answer set, by ranking objects with ranking scores, which is the advantage of the top-k query. The ranking function f(o) of any object o is defined as the number of objects dominated by object o in the data space. This ranking function can also achieve the advantages of the skyline query, that is, users do not have to specify a particular ranking function, and the results are independent of the attribute scales.

Top-k Dominating Query

- Given a spatial database D and an integer parameter k, a *top-k dominating query* returns k objects o in D with the highest score $f(o)$
  - $f(o)$ is the number of objects dominated by o



$f(p_4) = ?$

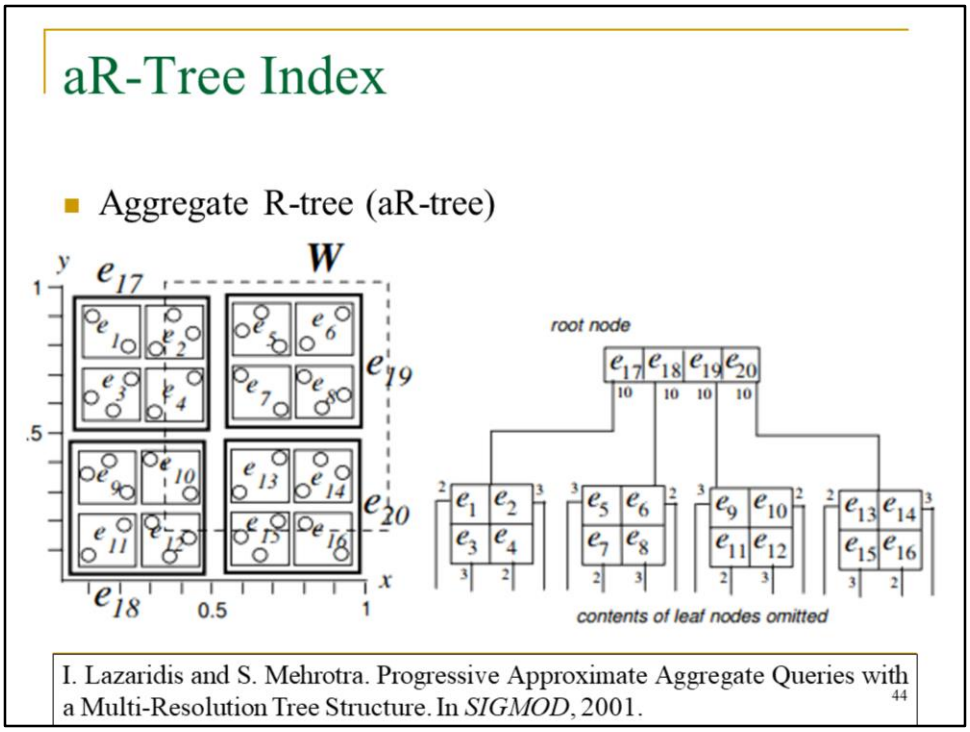
$f(p_5) = ?$

$k=2$

43

Formally, a top-k dominating query retrieves k objects from the database, such that these k objects have the highest ranking score, where the score function f(o) of an object o is given by the number of objects dominated by o.

As shown in the figure, object p_4 is dominating 3 objects p_5, p_2, and p_3, therefore, its ranking function f(p_4) outputs 3. Similarly, object p_5 dominates 2 objects p_2 and p_3, and its ranking score is 2. For other objects, their ranking scores are either 0 or 1. Thus, if user specifies parameter k = 2, then objects p_4 and p_5 will be returned as the top-k dominating query answers (as they have top-2 highest ranking scores).

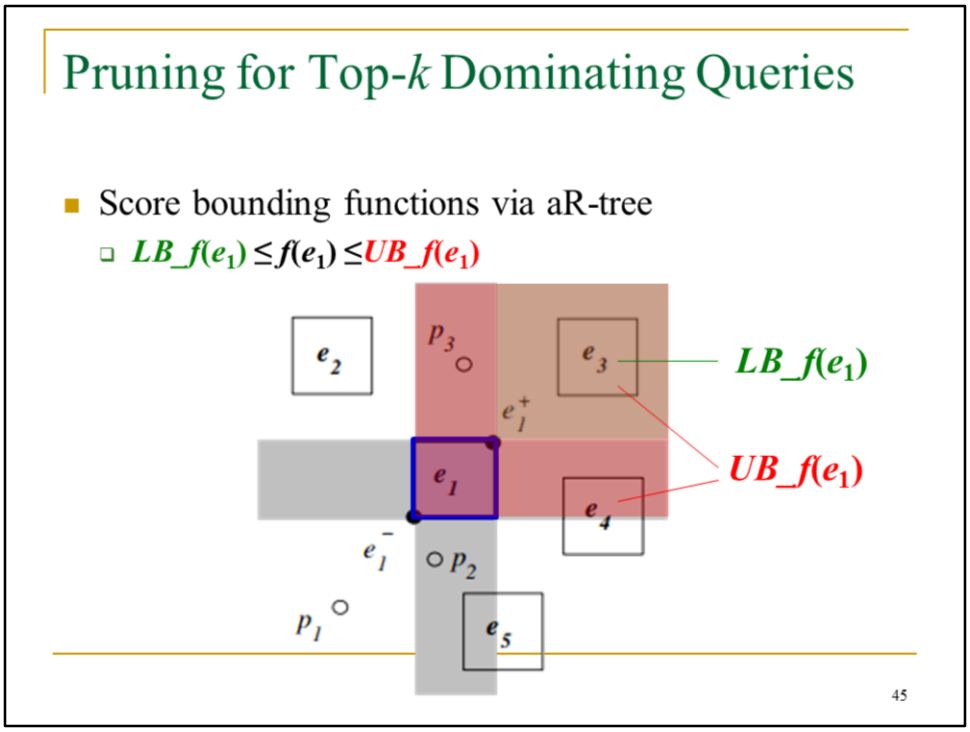


I. Lazaridis and S. Mehrotra. Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure. In *SIGMOD*, 2001.

D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient OLAP Operations in Spatial Data Warehouses. In SSTD, 2001.

M. L. Yiu and N. Mamoulis. Efficient Processing of Top-*k* Dominating Queries on Multi-Dimensional Data. In *VLDB*, 2007.

In order to count the number of objects dominated by an object, we can use aggregate R-tree (or aR-tree) index to facilitate the query processing. As we mentioned earlier, aR-tree stores aggregate information (e.g., COUNT) in the intermediate nodes, which can be used to estimate lower/upper bounds of objects dominated by an object.



Let us see an example on the node level of the aR-tree, where we have some node MBRs, e_1 ~ e_5. To estimate lower/upper bounds of the scores for any object inside node e_1, we can consider the top-right and bottom-down corners of MBR e_1, and obtain the numbers of objects dominated by these two corners.

In this example, MBR e_3 is dominated by top-right corner of e_1, and its COUNT aggregate can be considered as the score lower bound. MBRs e_3 and e_4 are fully and partially dominated by bottom-left corner of e_1, respectively, and their summed COUNT aggregate can be treated as the score upper bound.

Basic Pruning Idea

- Given two nodes e_1 and e_2
  - If $|e_1| \geq k$ and $UB\_f(e_2) < LB\_f(e_1)$, then node e_2 can be safely pruned

46

With the score lower/upper bounds of nodes, we can apply pruning methods when answering the top-k dominating query. Specifically, if the number of objects in node e_1 is greater than or equal to k, and the score upper bound of node e_2 is smaller than the score lower bound of node e_1, then node e_2 can be safely pruned. This is because there are at least k objects in e_1 having scores higher than that in e_2.

Outline

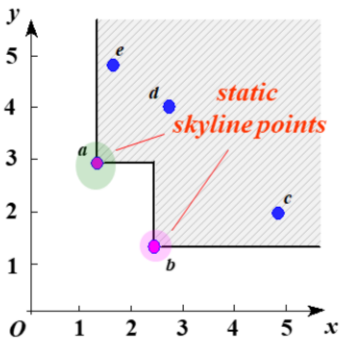
- Top-k Query
- Skyline Query
- Top-k Dominating Query
- Reverse Skyline Query
- Inverse Ranking Query

47

Similar to the relationship between nearest neighbor and reverse nearest neighbor, for the skyline query, we also have its counterpart, called reverse skyline query. Next, we will talk about this reverse skyline query.

Static Skyline Problem

- Point $o(o_1, o_2, \ldots, o_d)$ *dominates* point $p(p_1, p_2, \ldots, p_d)$, iff
  - $o_i \le p_i$ for *all* $1 \le i \le d$;
  - $o_j < p_j$, for *some* $1 \le j \le d$
- Point o is a *skyline point* if o is not *dominated* by other points

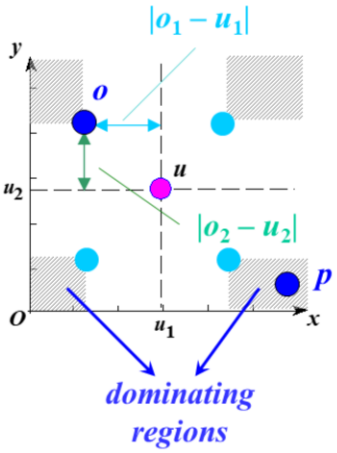


48

Recall that, for the skyline query, we want to find objects that are not dominated by others. If the dominance between two objects considers the static attributes (e.g., coordinates of object locations), we call such a skyline query static skyline.

Dynamic Skyline

- Skyline with dynamic attributes
- Dynamic dominance
  - $|o_i - u_i| \leq |p_i - u_i|$, for *all* $1 \leq i \leq d$
  - $|o_j - u_j| < |p_j - u_j|$, for *some* j
- To obtain all the objects in the database that are not *dynamically dominated* by other objects with respect to query object u



E. Dellis and B. Seeger. Efficient Computation of Reverse Skyline Queries. In *VLDB*, 2007.

49

In some application scenario, we may want to rank objects with respect to a reference point. For the hotel example of the skyline query, sometimes people have a rough idea about the price they can afford/want and distance to beach they desire. A cheap hotel usually indicates poor services. Thus, users can specify a reference hotel with desired hotel price and distance to the beach. Then, we can obtain all hotels with dynamic attributes w.r.t. this reference hotel, which are defined as the distance differences of attributes between each hotel and the reference hotel.

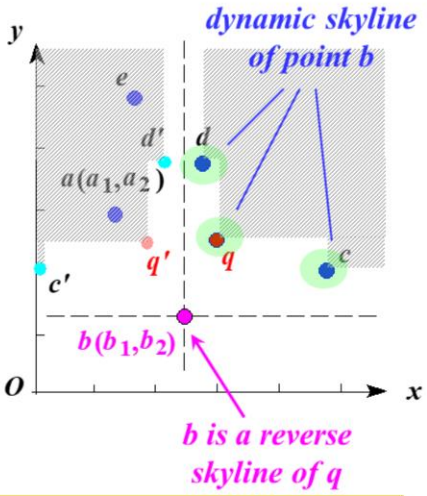
As shown in the figure, query point u is a reference query point, and other objects like o, have absolute distance differences between o and u on dimensions x and y, which correspond to dynamic attributes w.r.t. u.

With such dynamic attributes for each object, we can consider dynamic skyline. That is, given a query object u, a dynamic skyline query obtains all the objects in the database that are not dynamically dominated by other objects with respect to u.

As illustrated in the figure, with respect to reference point u and object o, we can define the dominating regions of object o in 4 quadrants of the 2D space. Any object p falling into the shaded dominating regions is dynamically dominated by object o (due to farther distances of p to reference point u on all dimensions).

Reverse Skyline Query

- Given a query point q, a *reverse skyline* query obtains all the objects u such that the dynamic skyline points of u include query point q



E. Dellis and B. Seeger. Efficient Computation of Reverse Skyline Queries. In *VLDB*, 2007.

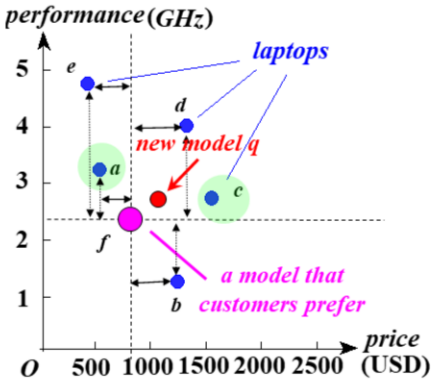
50

Now we will see how to define reverse skyline query (the inverse version of dynamic skyline query). Given a query point q, a reverse skyline query retrieves all objects u such that the dynamic skylines of u include the query point q.

As shown in the example of the figure, the dynamic skylines of object b include objects d, q, and c. Therefore, query point q is the dynamic skyline of b, and b is the reverse skyline of query point q.

Motivation Example of Reverse Skyline Query

- In a laptop market, each model corresponds to a 2D point in a *price*-and-*performance* space
- Those customers who are interested in *f*, are very likely to be interested in *a* and *c*
- If a company wants to produce a new model, …



E. Dellis and B. Seeger. Efficient Computation of Reverse Skyline Queries. In *VLDB*, 2007.

51

The reverse skyline has many real applications such as business planning. In a laptop market, each model of the laptop corresponds to a 2D point in a price-and-performance space. Intuitively, those customers who are interested in a laptop may also be interested in its dynamic skylines. For example, if a customer prefers laptop f, then he/she is also likely to be interested in its dynamic skylines a and c.

Now if a company wants to produce a new model q, we would like to see which customers will be interested in this new model q. In this case, we can issue a reverse skyline query with query point q, and obtain those customers who may also like this new model.

Problem Definition of Reverse Skyline Query

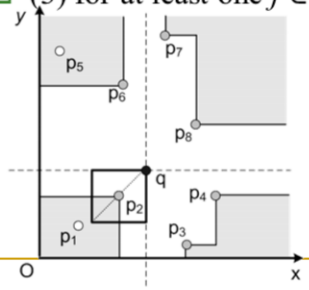
- Let P be a d-dimensional data set
- Given a query point q, a *reverse skyline query* (RSQ) according to the query point q retrieves all points $p_1 \in P$ where q is in the *dynamic skyline* of p_1.
- Formally, a point $p_1 \in P$ is a ***reverse skyline*** point of $q \in P$, iff $\neg\exists\, p_2 \in P$ such that:
  - (a) for *all* $i \in \{1, \ldots, d\}$: $|p_2{}^i - p_1{}^i| \leq |q^i - p_1{}^i|$ and
  - (b) for *at least one* $j \in \{1, \ldots, d\}$: $|\, p_2{}^j - p_1{}^j| < |\, q^j - p_1{}^j|$

52

Here is the formal definition of the reverse skyline query, which finds all objects whose dynamic skylines include the query point q.

Global Skyline Points

- A point $p_1 \in P$ *globally dominates* $p_2 \in P$ with regard to the query point q if:
  - (1) for all $i \in \{1, \ldots, d\}$: $(p_1^i - q^i)(p_2^i - q^i) > 0$,
  - (2) for all $i \in \{1, \ldots, d\}$: $|p_1^i - q^i| \leq |p_2^i - q^i|$, and
  - (3) for at least one $j \in \{1, \ldots, d\}$: $|p_1^j - q^j| < |p_2^j - q^j|$



The ***global skyline*** of a point q, GSL(q), contains those points which are not *globally dominated* by another point according to q

The global skyline set, GSL(q), is a superset of ***reverse skyline*** answers of query point q!!

53

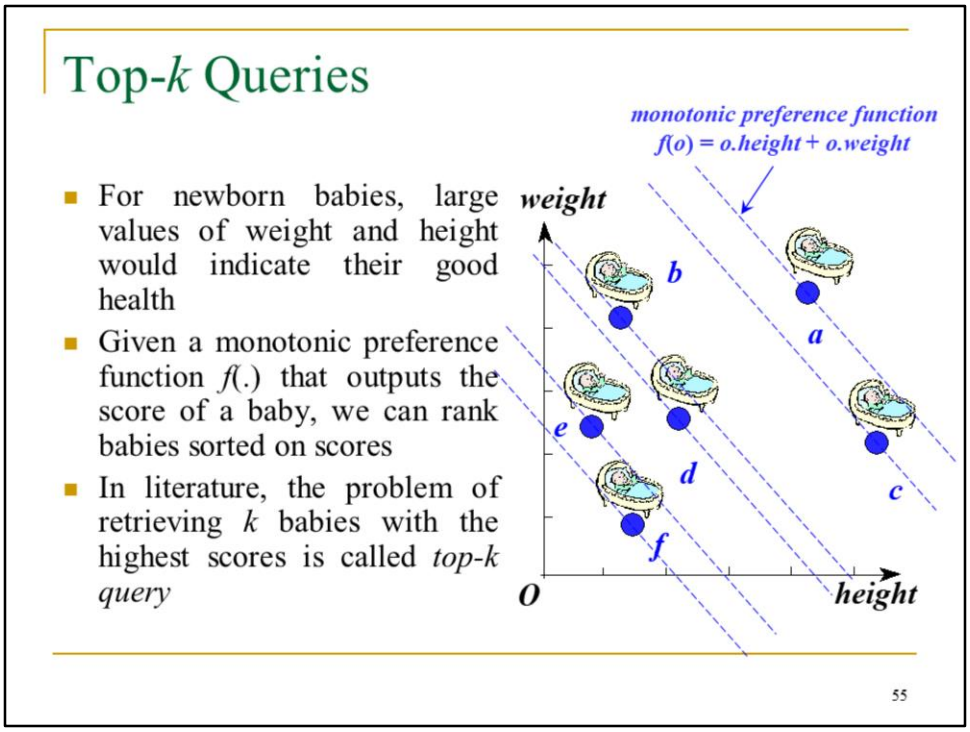
In the reverse skyline paper, they proposed efficient optimization techniques to retrieve reverse skyline points. One interesting property they found is the global skylines, which are skylines within each quadrant. It can be proved that the set of global skylines is actually a superset of reverse skyline answers. Therefore, for query processing, they only need to retrieve dynamic skylines within each quadrant of the data space.

Outline

- Top-*k* Query
- Skyline Query
- Top-*k* Dominating Query
- Reverse Skyline Query
- Inverse Ranking Query

54

Finally, we will talk about the inverse ranking query, which is related to the top-k query.



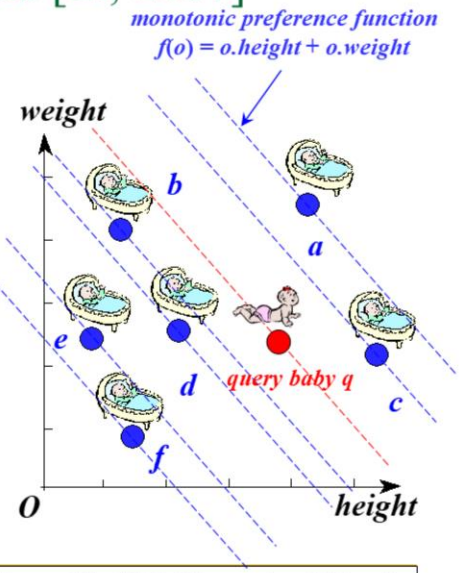
Let me first give an example of top-k query.

Assume that we have a number of newborn babies. Each baby has his/her own height and weight, which is represented by a point in this data space.

Normally, if a baby has large values of weight and height, then it indicates a good health of this baby. The top-k problem would specify a preference function, for example, to compute the summation of weight and height. Then, each baby has a score based on this preference function. In our example, baby a has the largest score, baby b has the second largest score, and so on. The top-k problem is to retrieve k babies who have the highest scores, that is, k healthiest babies.

Inverse Ranking Queries [Li, 2007]

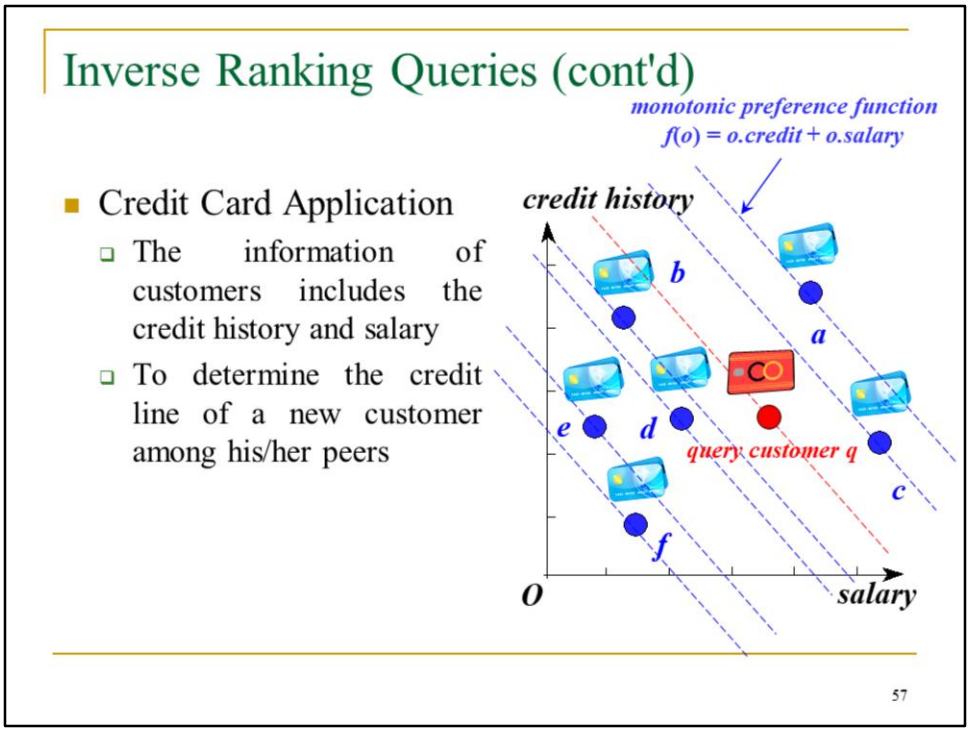
- For a newborn baby, we may be interested in his/her health compared with other babies, for example, in terms of height and weight
- Given a query baby q and a preference function $f(.)$, an *inverse ranking* problem is to obtain the rank of baby q among other babies, in order to infer q's health



C. Li. Enabling data retrieval: by ranking and beyond. In *Ph.D. Dissertation, University of Illinois at Urbana-Champaign*, 2007.

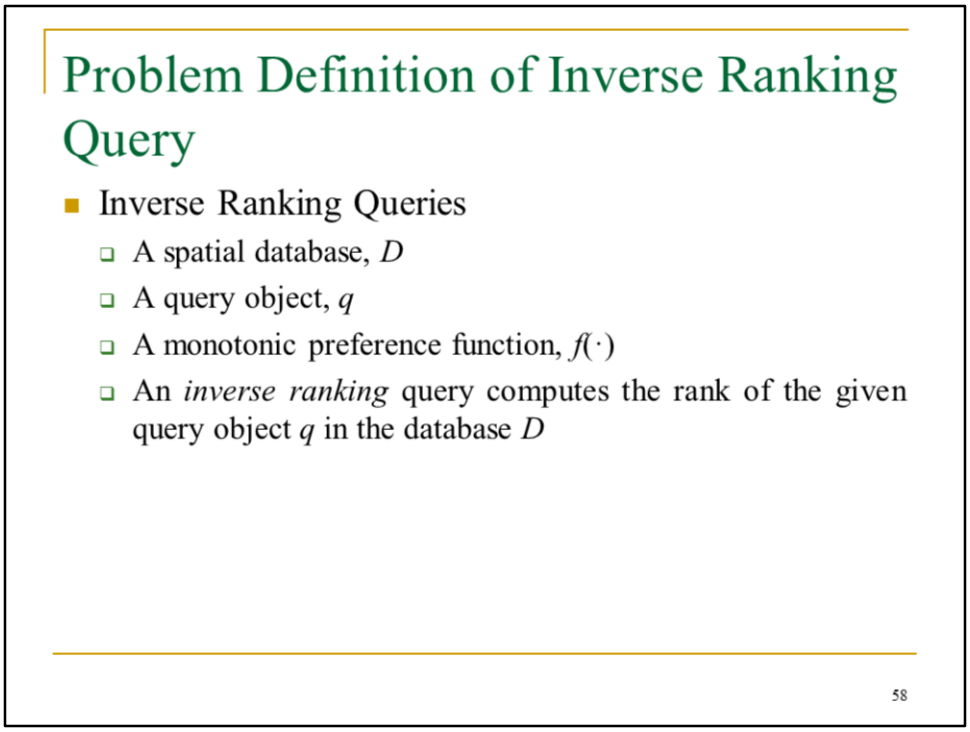
56

In contrast, for the inverse ranking query, instead of finding babies with the highest scores in the existing database, we are given a query baby *q*, and we want to determine the health level of this query baby compared with other babies. In other words, we want to obtain the rank of this query baby in this database. In this example, we can see that the rank of this query baby is 3.



The inverse ranking query has many other applications, for example, in the credit card application. Each customer has his/her own information including the credit history and salary. Similarly, assume that we have a preference function which sums up credit and salary of each customer.

Now given a new customer q, the credit card company may want to determine the credit line of this customer by comparing his/her score with those of other customers. In this case, we can use the inverse ranking query to obtain the rank of this new customer q among all other customers.

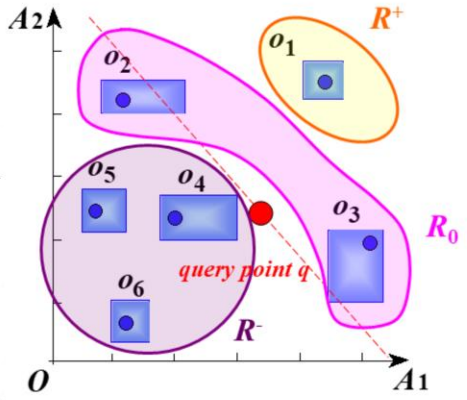


Here is the formal definition of the inverse ranking query. Given a spatial database D, a query object q, and a preference function f(), an inverse ranking query computes the rank of the query object q in the database D.

Definitely it is not efficient to compute the scores for all objects in the database, sort them according to scores, and obtain the rank of the query object q.

Reducing the Search Space

- Classify MBRs/objects into 3 categories
  - R^+ – a set of MBRs/objects o with scores $f(o)$ *definitely larger than* $f(q)$
  - R_0 – a set of MBRs/objects o that *might* have scores $f(o)$ *equal to* $f(q)$
  - R^- – a set of MBRs/objects o with scores $f(o)$ *definitely smaller than* $f(q)$



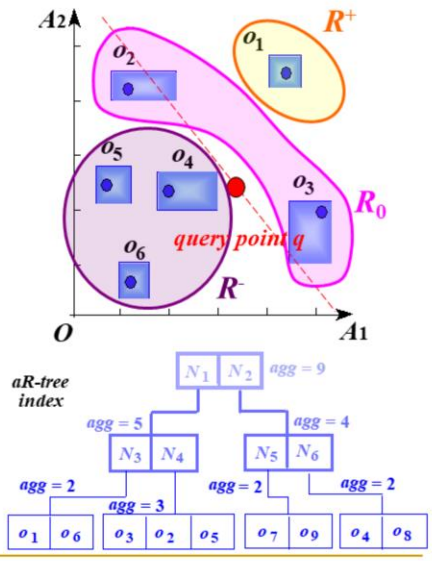
59

To efficiently tackle this problem, we can use a variant of the R-tree index, aR-tree, to facilitate the query processing. In particular, in order to compute the rank of the query object q, we can classify MBRs or objects in the data space into 3 categories (or subsets), R+, R_0, and R-. For R+, scores of MBRs/objects are definitely larger than that of query object q. For R-, scores of MBRs/objects are definitely smaller than that of q. Moreover, MBRs/objects in R_0 may have scores smaller or greater than that of q.

By such a classification, our inverse ranking query only needs to check objects in sets R+ and R_0, and see how many objects have scores exactly greater than that of q.

Pruning Idea

- Indexing
  - Aggregate R-tree (aR-tree)
  - Aggregate: COUNT
- While traversing aR-tree
  - Obtain the size, $|R^+|$, of set R^+
  - Retrieve all the MBRs /objects in the set R_0



60

We use aR-tree index which stores the COUNT aggregate information in MBR nodes. During the index traversal, we only need to obtain the size of subset R+, without the need to access the detailed objects under the MBR nodes. Moreover, for subset R_0, we need to traverse to the leaf node level, and count objects with higher scores than f(q). This way, we can efficiently retrieve the answers to the inverse ranking query.