Skyline queries and its variations

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

Presented by Jagan Sankaranarayanan

Skyline Queries

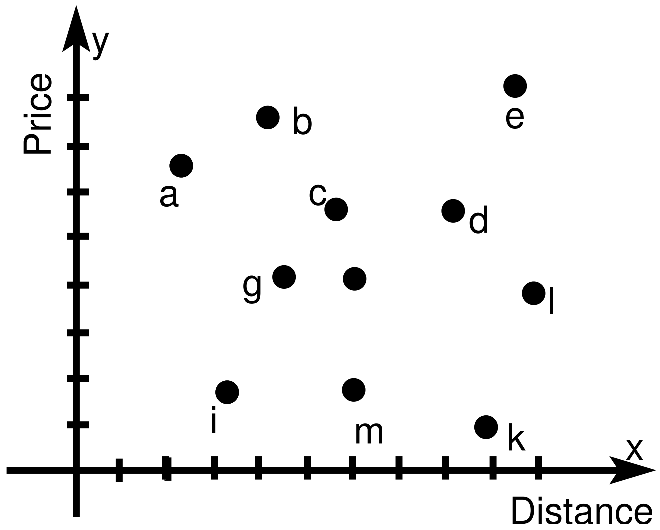
- Definition: Given a set of points $p_1, p_2, ...p_N$, the skyline query returns a set of points P (referred to as the *skyline points*), such that any point $p_i \in P$ is not dominated by any other point in the dataset.

Skyline Queries

- Definition: Given a set of points $p_1, p_2, ...p_N$, the skyline query returns a set of points P (referred to as the *skyline points*), such that any point $p_i \in P$ is not dominated by any other point in the dataset.
- Definition of point domination: a point p_i *dominates* another point p_j if and only if the coordinate of p_i on any axis is not *larger* than the corresponding coordinate of p_j
- Informally, *larger* translates to an *preference function* that is a monotone on all attributes.

Example

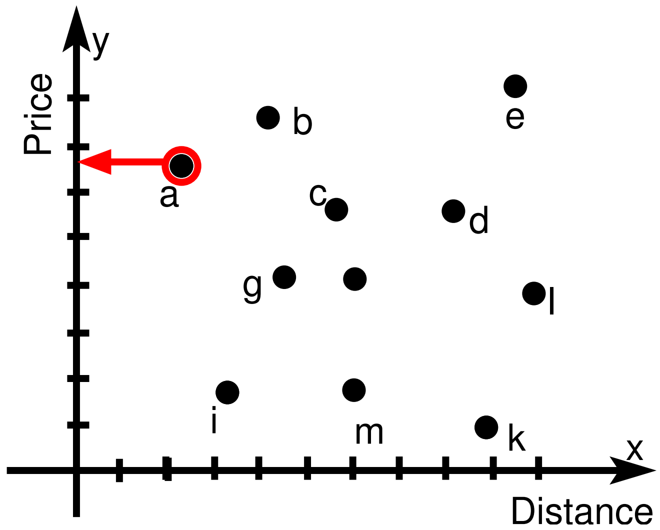
- A dataset containing information about hotels; the *distance* to the beach and the *price* for each data point is recorded.
- Consider a two dimensional plot of the dataset, where the distance and price are assigned to the X,Y axis of the plot.



- the goal of the search is to find a hotel whose distance to the beach and the price are both minimum (not restricted to minimum, any other function max, join, group-by clause could be used.)
- the preference function in our example is "minimum price and minimum distance". The dataset may not have one single data point that satisfies both these desirable properties.
- the user is presented with a set of interesting points that partly satisfy the imposed constraints.

Example

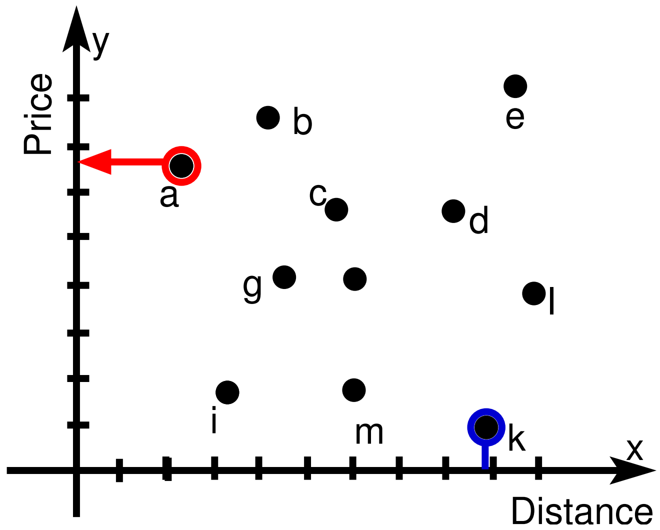
- A dataset containing information about hotels; the *distance* to the beach and the *price* for each data point is recorded.
- Consider a two dimensional plot of the dataset, where the distance and price are assigned to the X,Y axis of the plot.



- The interesting data-points in the dataset are $\{a, i, k\}$. a has the least distance from the beach

Example

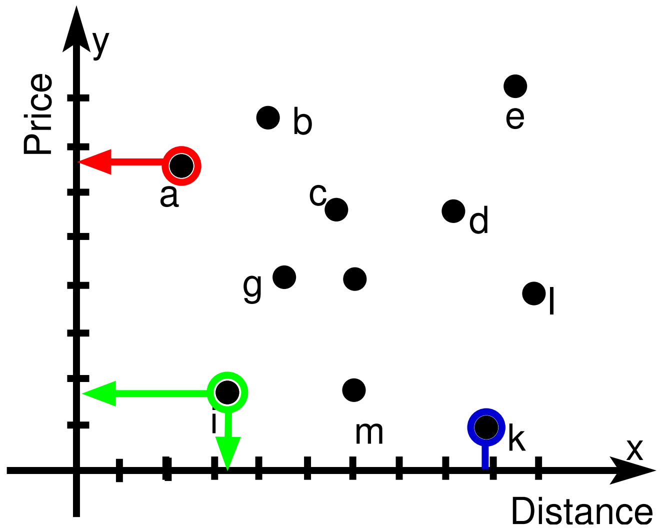
- A dataset containing information about hotels; the *distance* to the beach and the *price* for each data point is recorded.
- Consider a two dimensional plot of the dataset, where the distance and price are assigned to the X,Y axis of the plot.



- The interesting data-points in the dataset are $\{a, i, k\}$. a has the least distance from the beach , k has the lowest price

Example

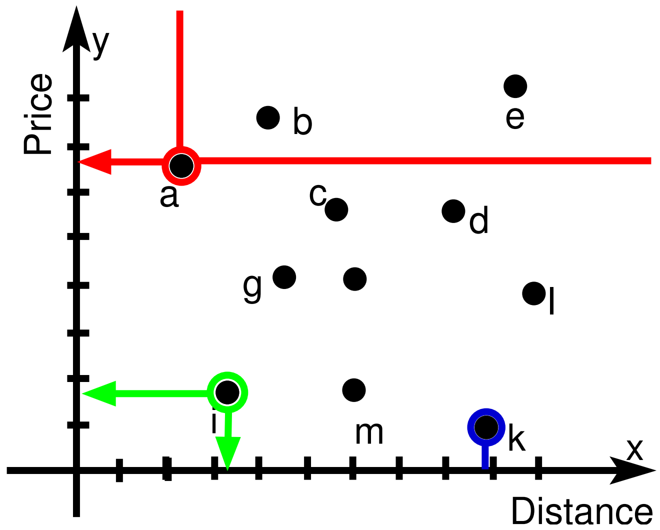
- A dataset containing information about hotels; the *distance* to the beach and the *price* for each data point is recorded.
- Consider a two dimensional plot of the dataset, where the distance and price are assigned to the X,Y axis of the plot.



- The interesting data-points in the dataset are $\{a, i, k\}$. a has the least distance from the beach , k has the lowest price , i has neither the shortest distance nor the minimum price.

Example

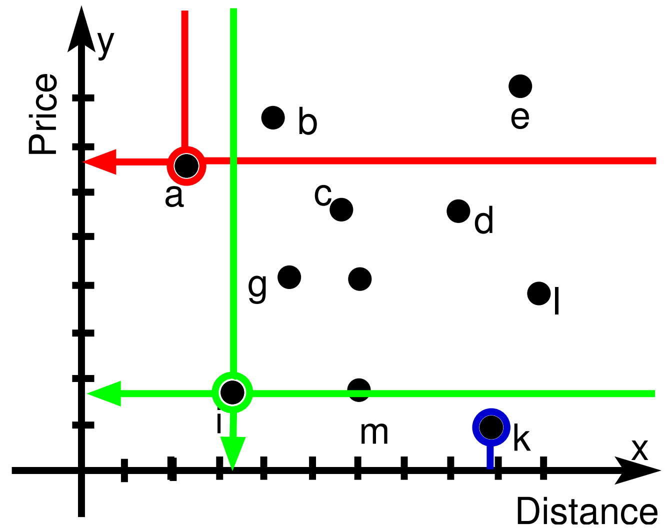
- A dataset containing information about hotels; the *distance* to the beach and the *price* for each data point is recorded.
- Consider a two dimensional plot of the dataset, where the distance and price are assigned to the X,Y axis of the plot.



- The interesting data-points in the dataset are $\{a, i, k\}$. a has the least distance from the beach , k has the lowest price , i has neither the shortest distance nor the minimum price. i has a lesser distance value than k and a lower price value than a. i is hence not dominated by either one of them.

Example

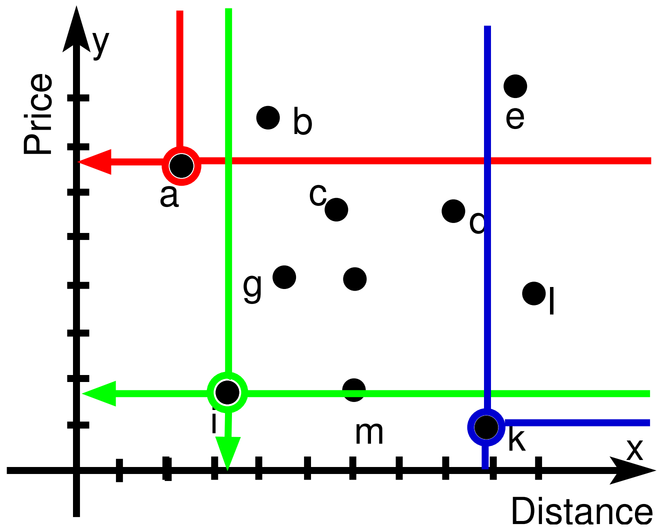
- A dataset containing information about hotels; the *distance* to the beach and the *price* for each data point is recorded.
- Consider a two dimensional plot of the dataset, where the distance and price are assigned to the X,Y axis of the plot.



- The interesting data-points in the dataset are $\{a, i, k\}$. a has the least distance from the beach , k has the lowest price , i has neither the shortest distance nor the minimum price. i has a lesser distance value than k and a lower price value than a. i is hence not dominated by either one of them.
- All other points in the dataset are dominated by the set of points $\{a, i, k\}$, i.e., both the distance and price values are greater than one or more skyline points

Example

- A dataset containing information about hotels; the *distance* to the beach and the *price* for each data point is recorded.
- Consider a two dimensional plot of the dataset, where the distance and price are assigned to the X,Y axis of the plot.



- The interesting data-points in the dataset are $\{a, i, k\}$. a has the least distance from the beach , k has the lowest price , i has neither the shortest distance nor the minimum price. i has a lesser distance value than k and a lower price value than a. i is hence not dominated by either one of them.
- All other points in the dataset are dominated by the set of points $\{a, i, k\}$, i.e., both the distance and price values are greater than one or more skyline points

Related algorithm

- convex hulls: contain the subset of skyline queries
- top-K queries: if the preference function is formulated as a cost-minimization function, Top-K queries retrieves skyline points.
- related to multivariate optimization, maximum vectors and contour problems.

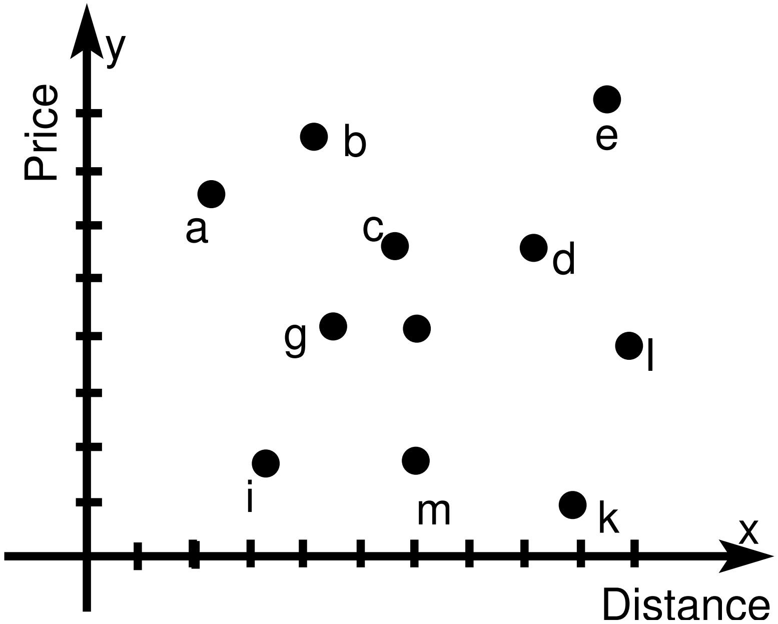
Techniques for evaluating skyline queries

- Block Nested Loop
- Divide and Conquer
- Plane-sweep
- Nearest Neighbor Search
- Branch and Bound Skyline

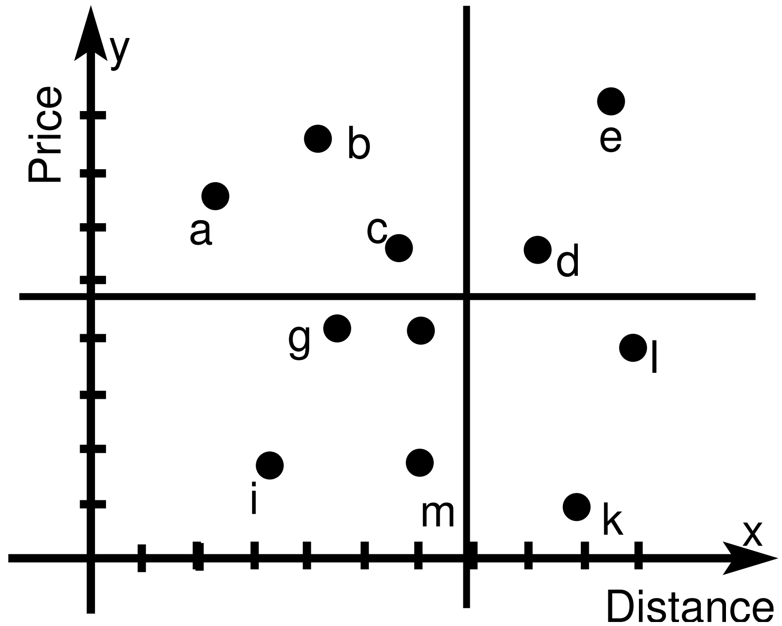
Block Nested Loop [Borzyoni,ICDE-2001]

- scan through a list of point and test each point for dominance criteria.
- active list of potential skyline points seen thus far are maintained, each visited point is compared with all elements in the list. The list is suitably updated.
- method does not require a precomputed index. Execution independent of the dimensionality of the space.
- total work done depends on the order in which points were encountered. method performs redundant work, no provision for early termination.

Divide-and-Conquer [Borzyoni,ICDE-2001]

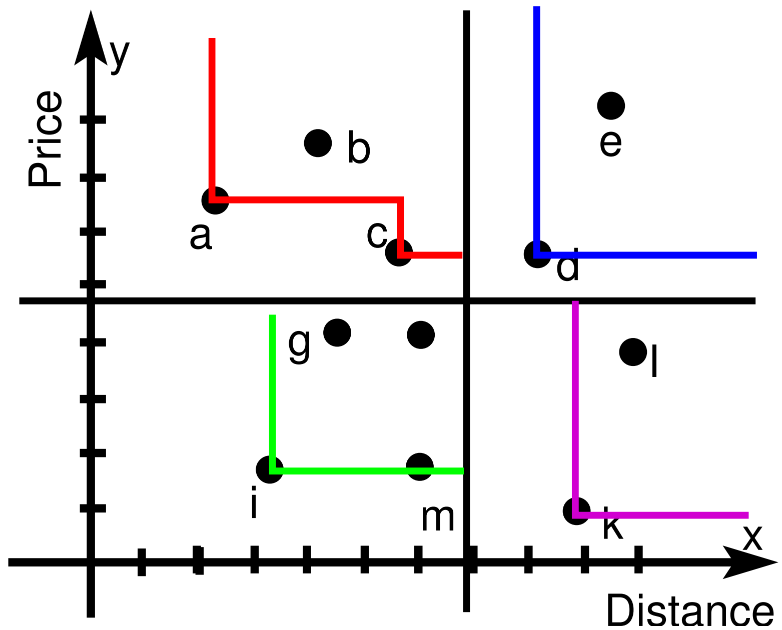


Divide-and-Conquer [Borzyoni,ICDE-2001]



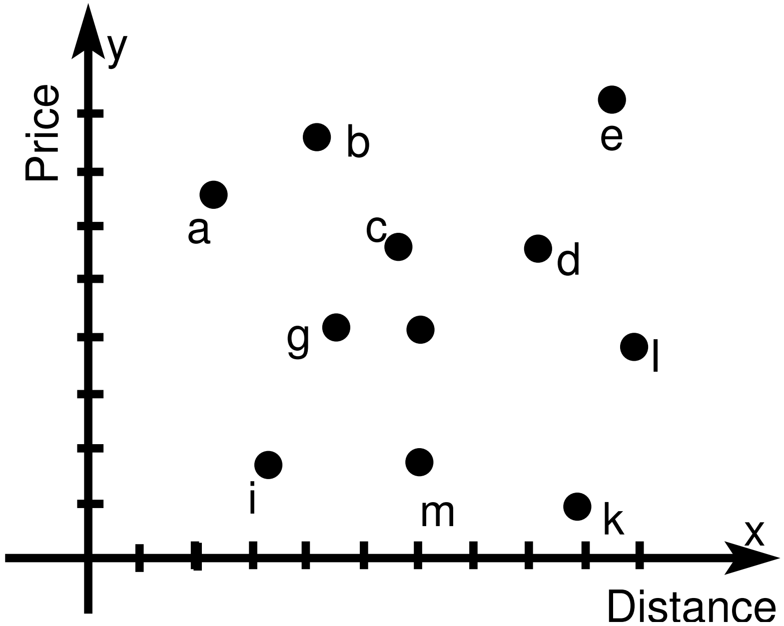
- recursively break up large datasets into smaller partition. Continue till each smaller partition of the dataset fits in the main memory.

Divide-and-Conquer [Borzyoni,ICDE-2001]



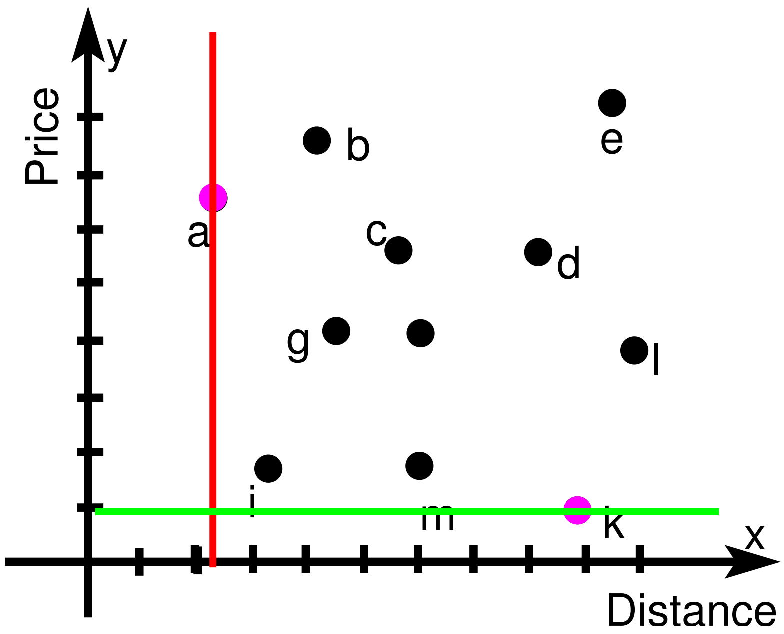
- recursively break up large datasets into smaller partition. Continue till each smaller partition of the dataset fits in the main memory.
- compute the partial skyline for each partition using any in-memory approach and later combine these partial skyline points to form the final skyline query.

Plane-sweep [Tan,VLDB-2001]



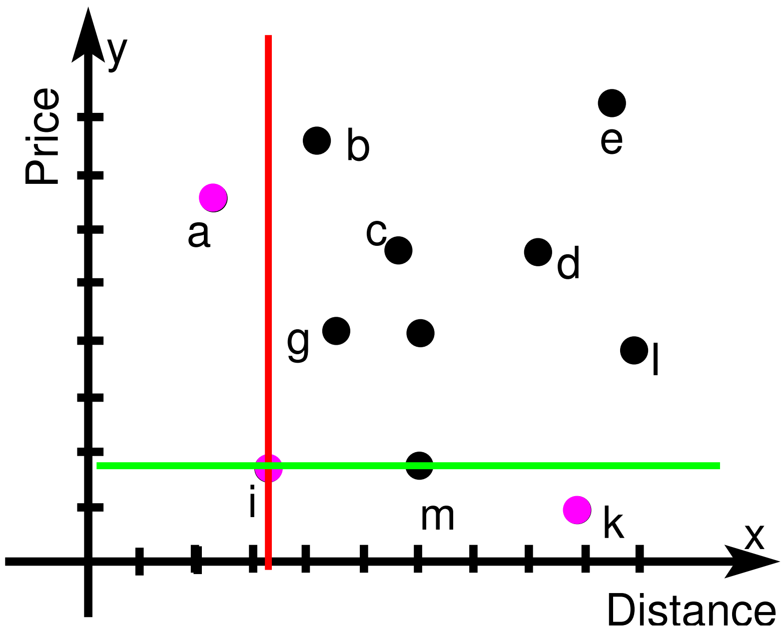
- plane sweep along each of the d dimensions.
- can detect early termination

Plane-sweep [Tan,VLDB-2001]



- plane sweep along each of the d dimensions.
- can detect early termination

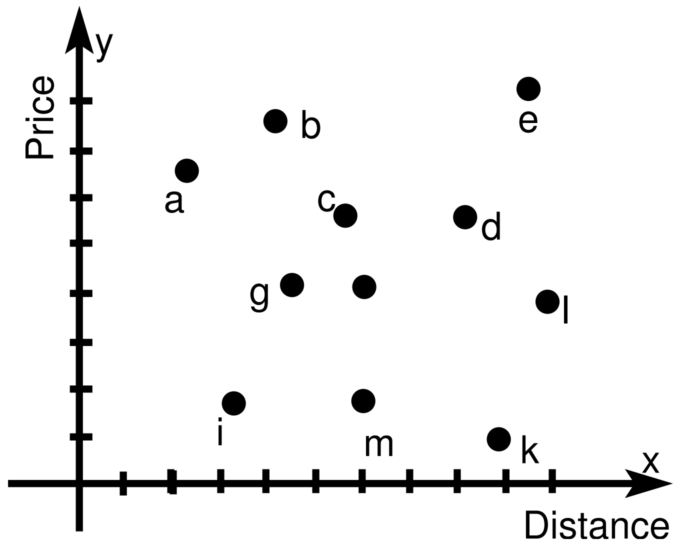
Plane-sweep [Tan,VLDB-2001]



- plane sweep along each of the d dimensions.
- can detect early termination

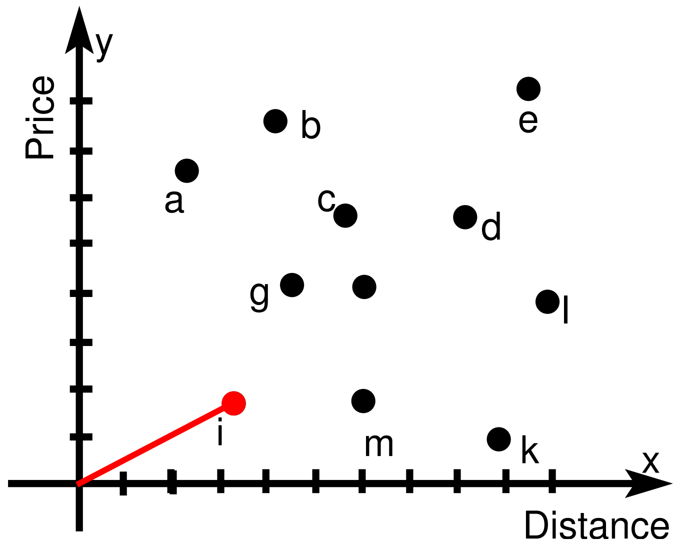
Nearest Neighbor Search [Kossman,VLDB-2002]

- assumes that a spatial index structure on the data points is available for use.
- identifies skyline points by repeated application of a nearest neighbor search technique on the data points, using a suitably defined L_1 distance norm.



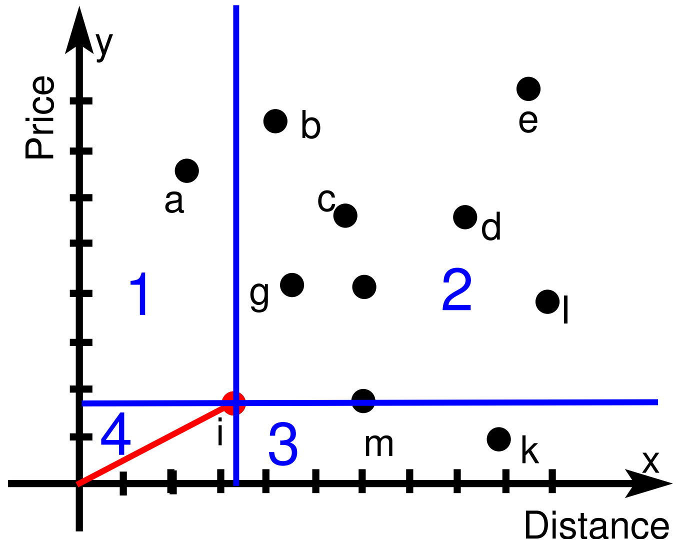
Nearest Neighbor Search [Kossman,VLDB-2002]

- assumes that a spatial index structure on the data points is available for use.
- identifies skyline points by repeated application of a nearest neighbor search technique on the data points, using a suitably defined L_1 distance norm.
- the nearest neighbor in the data-point is i as it is closest to the origin when an L_1 distance measure is assumed.



Nearest Neighbor Search [Kossman,VLDB-2002]

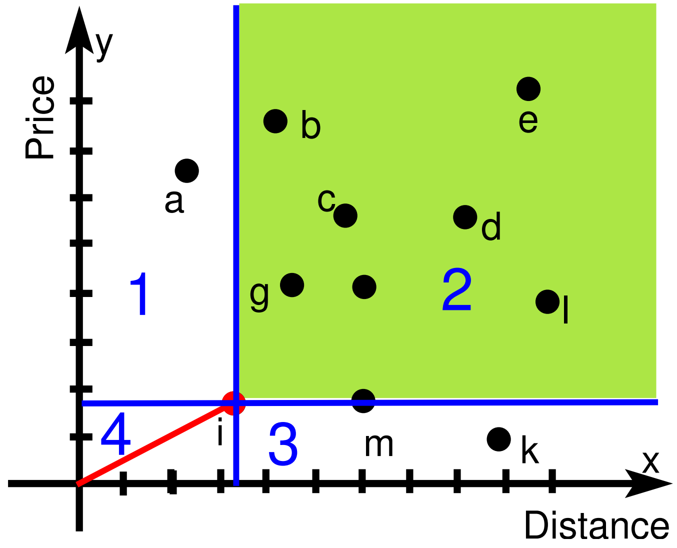
- assumes that a spatial index structure on the data points is available for use.
- identifies skyline points by repeated application of a nearest neighbor search technique on the data points, using a suitably defined L_1 distance norm.
- the nearest neighbor in the data-point is i as it is closest to the origin when an L_1 distance measure is assumed.



- i divides the space into 2^d non-disjoint region, which now must now be recursively searched for more skyline points.

Nearest Neighbor Search [Kossman,VLDB-2002]

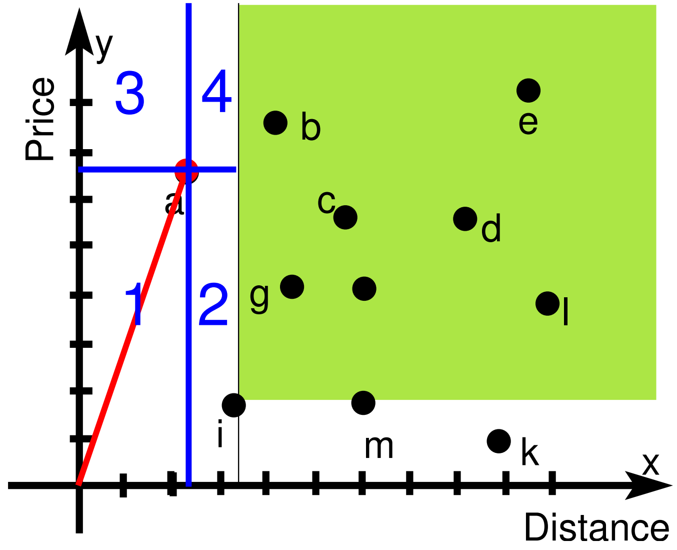
- assumes that a spatial index structure on the data points is available for use.
- identifies skyline points by repeated application of a nearest neighbor search technique on the data points, using a suitably defined L_1 distance norm.
- the nearest neighbor in the data-point is i as it is closest to the origin when an L_1 distance measure is assumed.



- However, region 4 and 2 need not be searched. The rest of the $2^d - 2$ regions need to be searched
  - No closer point than i in 2 (by virtue that i is the nearest neighbor to the origin). Any data-point in the space 2 is dominated by i.

Nearest Neighbor Search [Kossman,VLDB-2002]

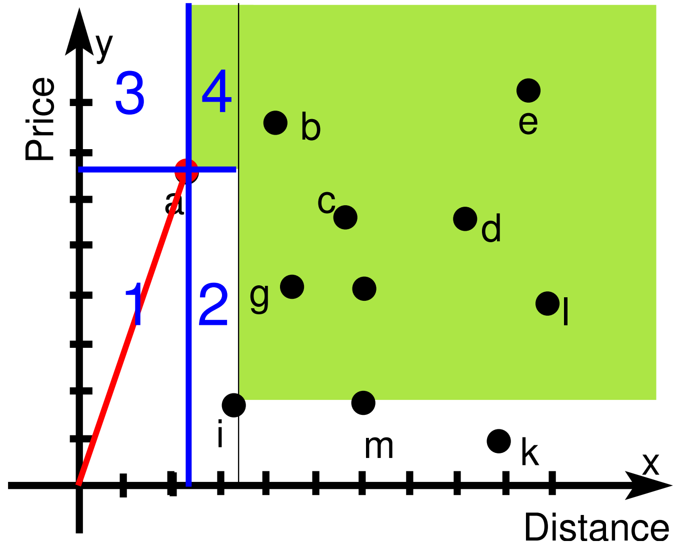
- assumes that a spatial index structure on the data points is available for use.
- identifies skyline points by repeated application of a nearest neighbor search technique on the data points, using a suitably defined L_1 distance norm.
- the nearest neighbor in the data-point is i as it is closest to the origin when an L_1 distance measure is assumed.



- However, region 4 and 2 need not be searched. The rest of the $2^d - 2$ regions need to be searched
  - No closer point than i in 2 (by virtue that i is the nearest neighbor to the origin). Any data-point in the space 2 is dominated by i.
- recursively apply the search on region-1. The nearest neighbor in region-1 would be a, explode region to form additional regions

Nearest Neighbor Search [Kossman,VLDB-2002]

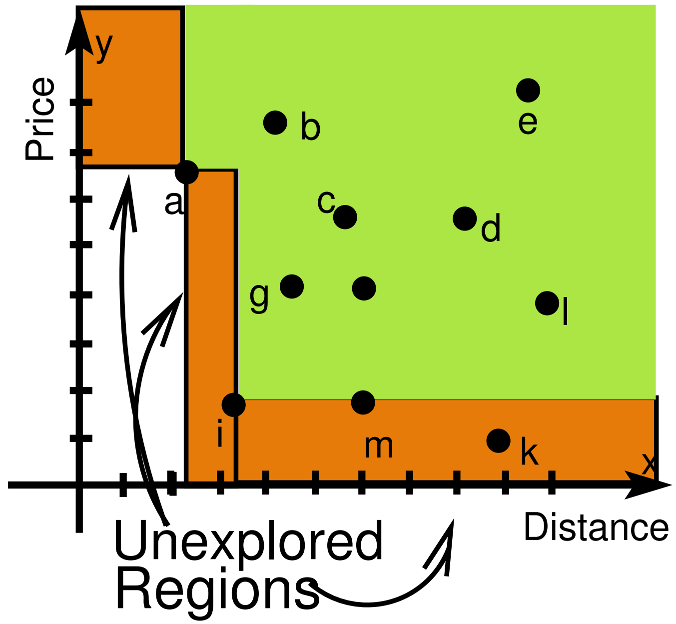
- assumes that a spatial index structure on the data points is available for use.
- identifies skyline points by repeated application of a nearest neighbor search technique on the data points, using a suitably defined L_1 distance norm.
- the nearest neighbor in the data-point is i as it is closest to the origin when an L_1 distance measure is assumed.



- However, region 4 and 2 need not be searched. The rest of the $2^d - 2$ regions need to be searched
  - No closer point than i in 2 (by virtue that i is the nearest neighbor to the origin). Any data-point in the space 2 is dominated by i.
- recursively apply the search on region-1. The nearest neighbor in region-1 would be a, explode region to form additional regions
- region-4 is added to the *pruned* region and need not be searched

Nearest Neighbor Search [Kossman,VLDB-2002]

- assumes that a spatial index structure on the data points is available for use.
- identifies skyline points by repeated application of a nearest neighbor search technique on the data points, using a suitably defined L_1 distance norm.
- the nearest neighbor in the data-point is i as it is closest to the origin when an L_1 distance measure is assumed.



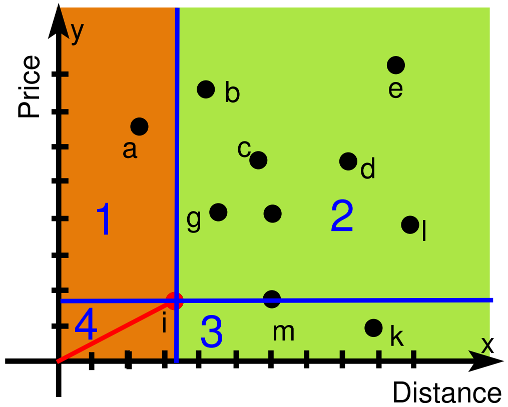
- However, region 4 and 2 need not be searched. The rest of the $2^d - 2$ regions need to be searched
  - No closer point than i in 2 (by virtue that i is the nearest neighbor to the origin). Any data-point in the space 2 is dominated by i.
- recursively apply the search on region-1. The nearest neighbor in region-1 would be a, explode region to form additional regions
- region-4 is added to the *pruned* region and need not be searched
- the number of *unexplored regions* grow rapidly - $O(dataset)$. The non-disjoint condition is relaxed for high-dimensional datasets.

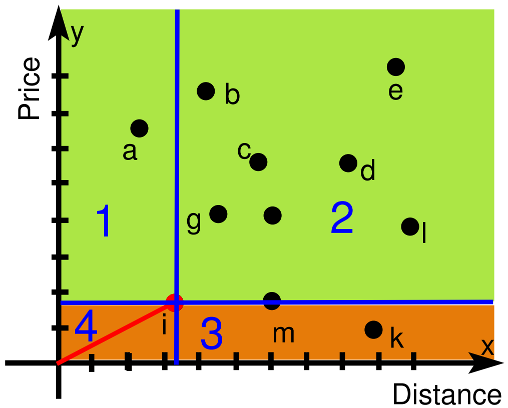
Overlapping the search regions

- relax the restriction that regions are non-overlapping. Assume that the point query splits each dimension into two regions; instead of exploding a region to 2^d, it reduces to $2d$.
- we have traded lesser number of regions to search at the expense dealing with duplicate and their removal.

Overlapping the search regions

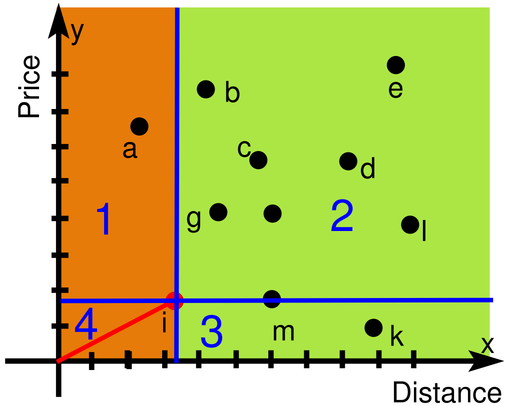
- relax the restriction that regions are non-overlapping. Assume that the point query splits each dimension into two regions; instead of exploding a region to 2^d, it reduces to $2d$.
- we have traded lesser number of regions to search at the expense dealing with duplicate and their removal.

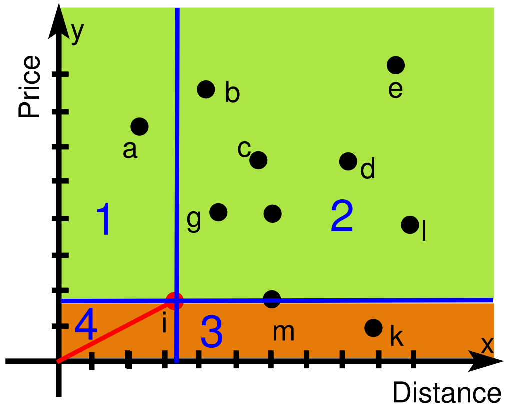




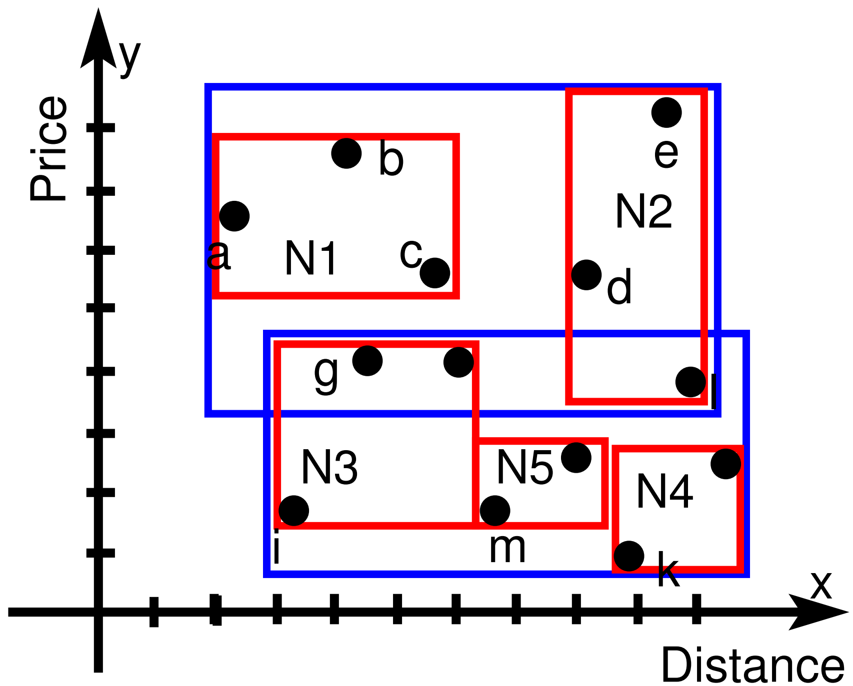
Overlapping the search regions

- relax the restriction that regions are non-overlapping. Assume that the point query splits each dimension into two regions; instead of exploding a region to 2^d, it reduces to $2d$.
- we have traded lesser number of regions to search at the expense dealing with duplicate and their removal.
- any one of these duplicate removal technique can be employed
  1. Laisser-Faire: maintain a in-memory hash table that keys in each point and flags it a duplicate if already present in the hash-table.
  2. Propagate: when a point p is found, remove all instances of p from all unvisited nodes.
  3. Merge: merge partitions to form non-overlap regions.



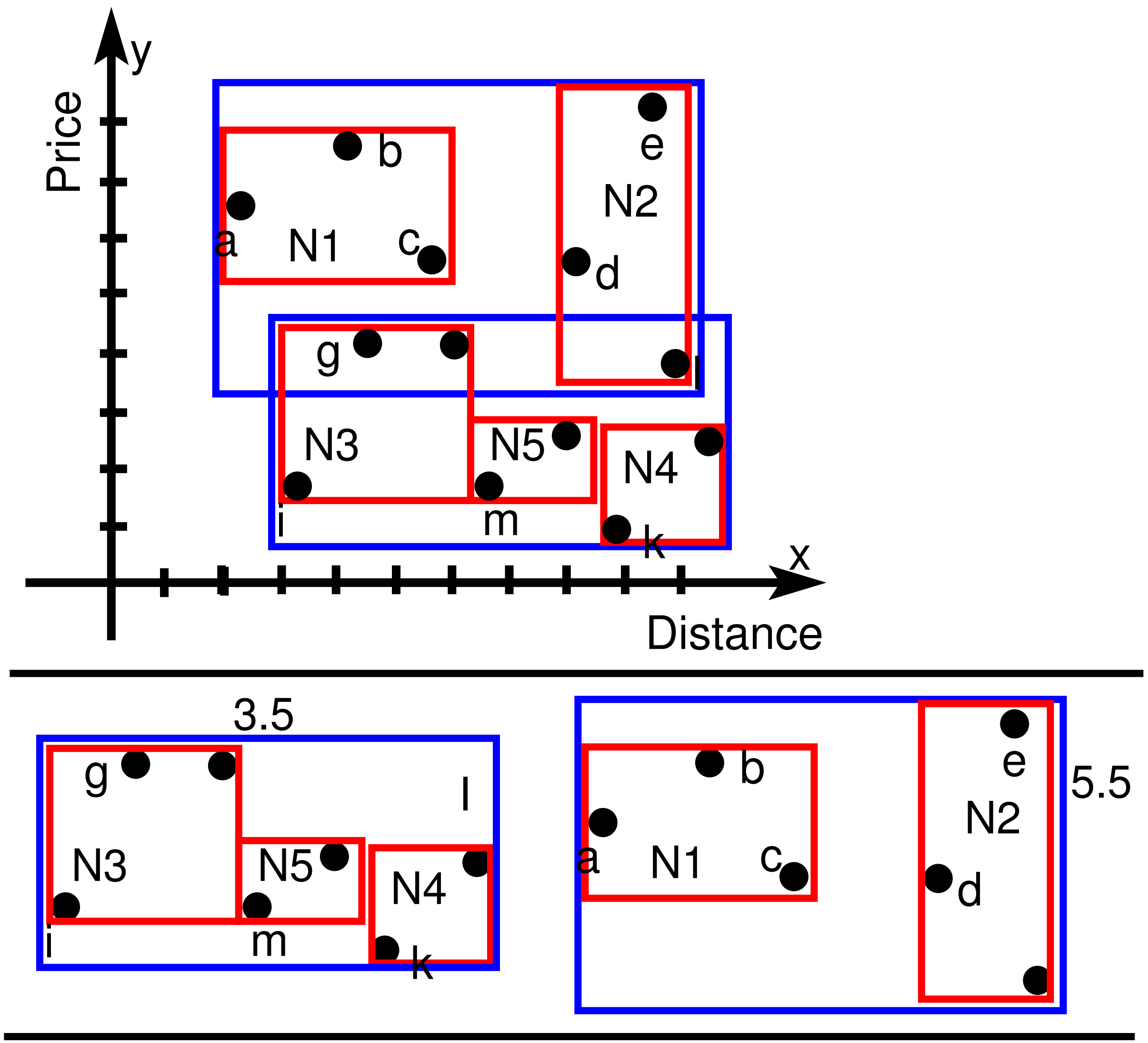


Branch and Bound Skyline [Papadias, SIGMOD-2003]



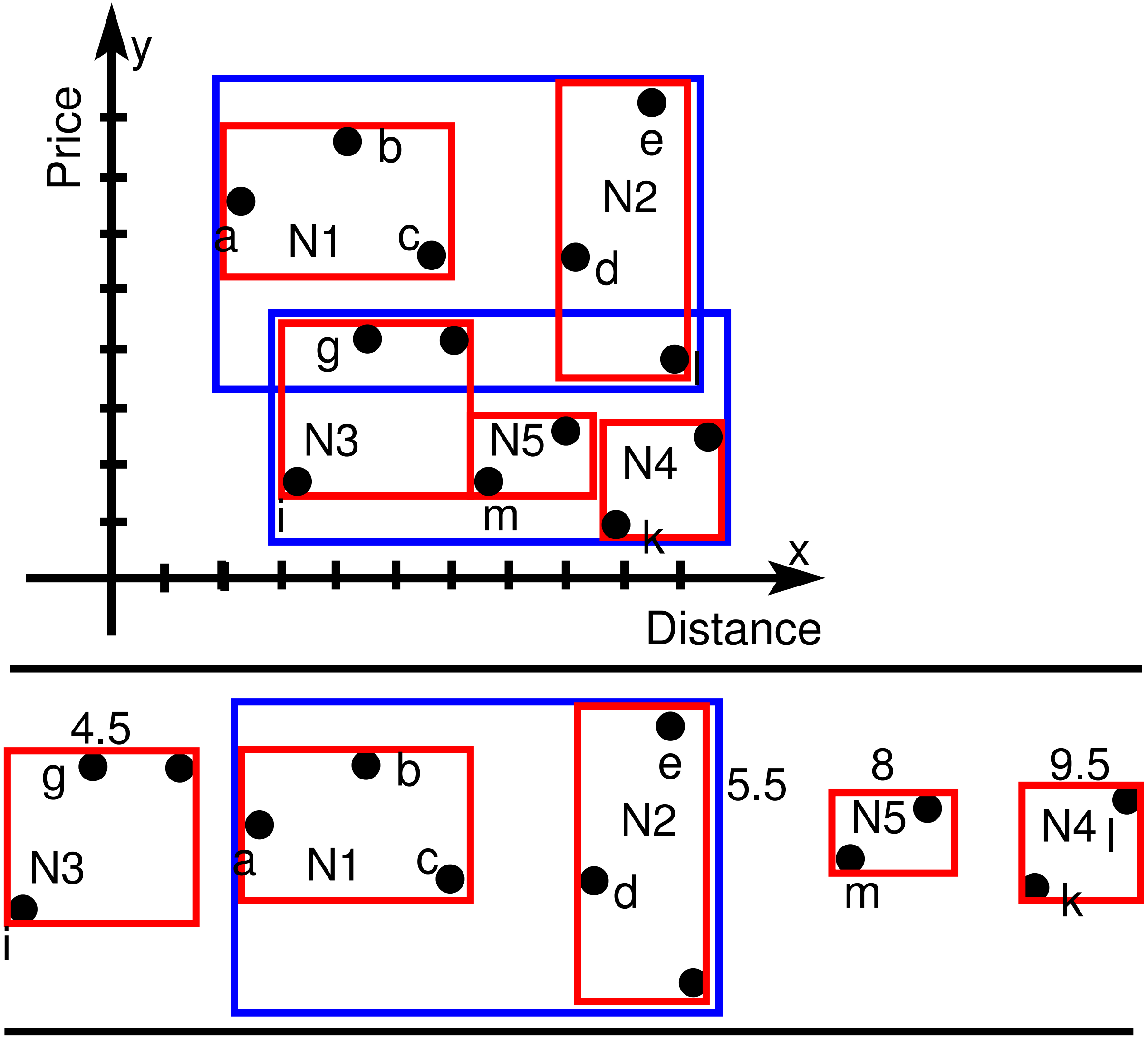
- an R-tree is built on the data points. Construct a priority queue that arranges objects in an MinDist ordering relative to the origin (uses an L_1 distance norm)

Branch and Bound Skyline [Papadias, SIGMOD-2003]



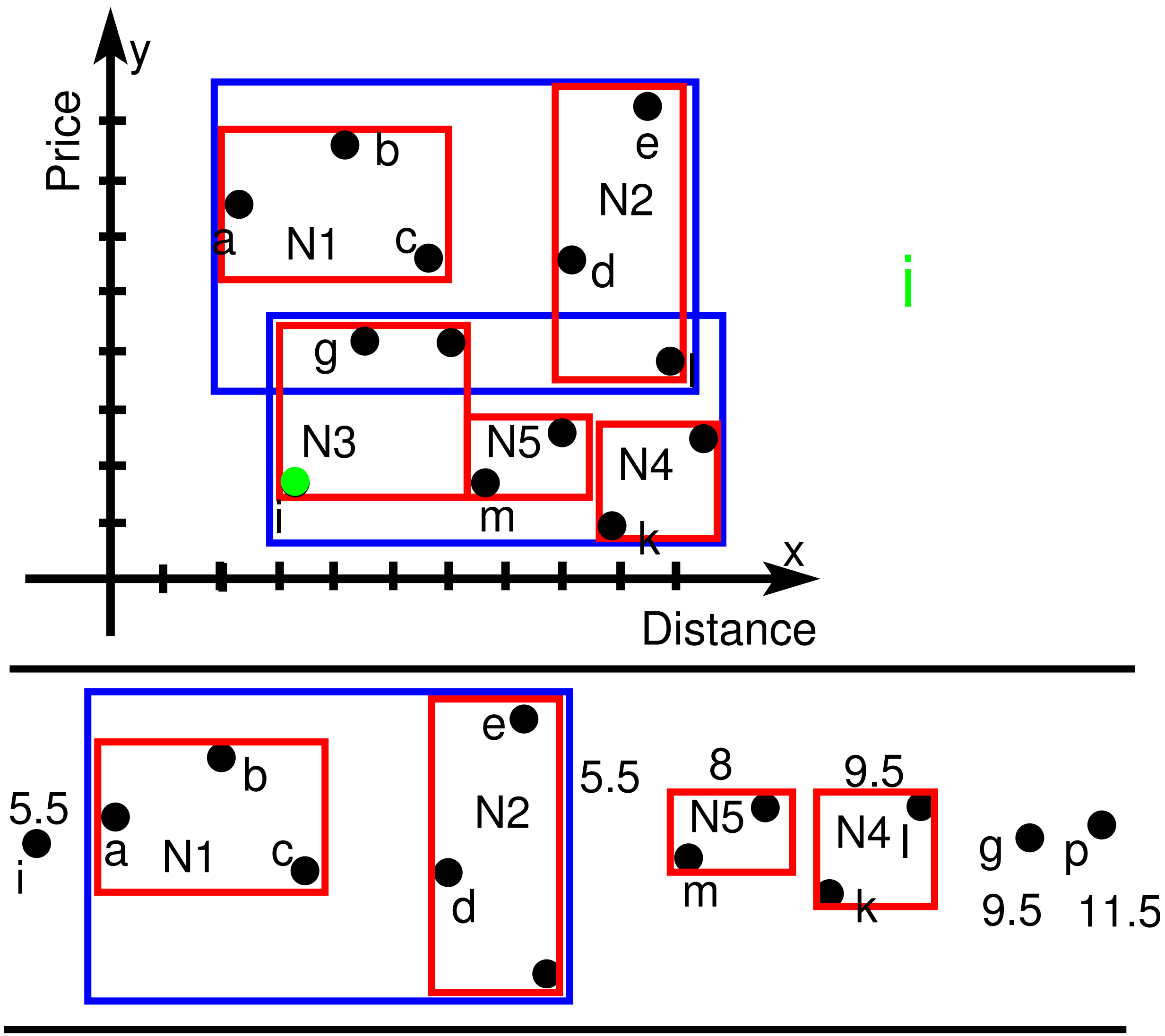
- insert top level of the hierarchy

Branch and Bound Skyline [Papadias, SIGMOD-2003]



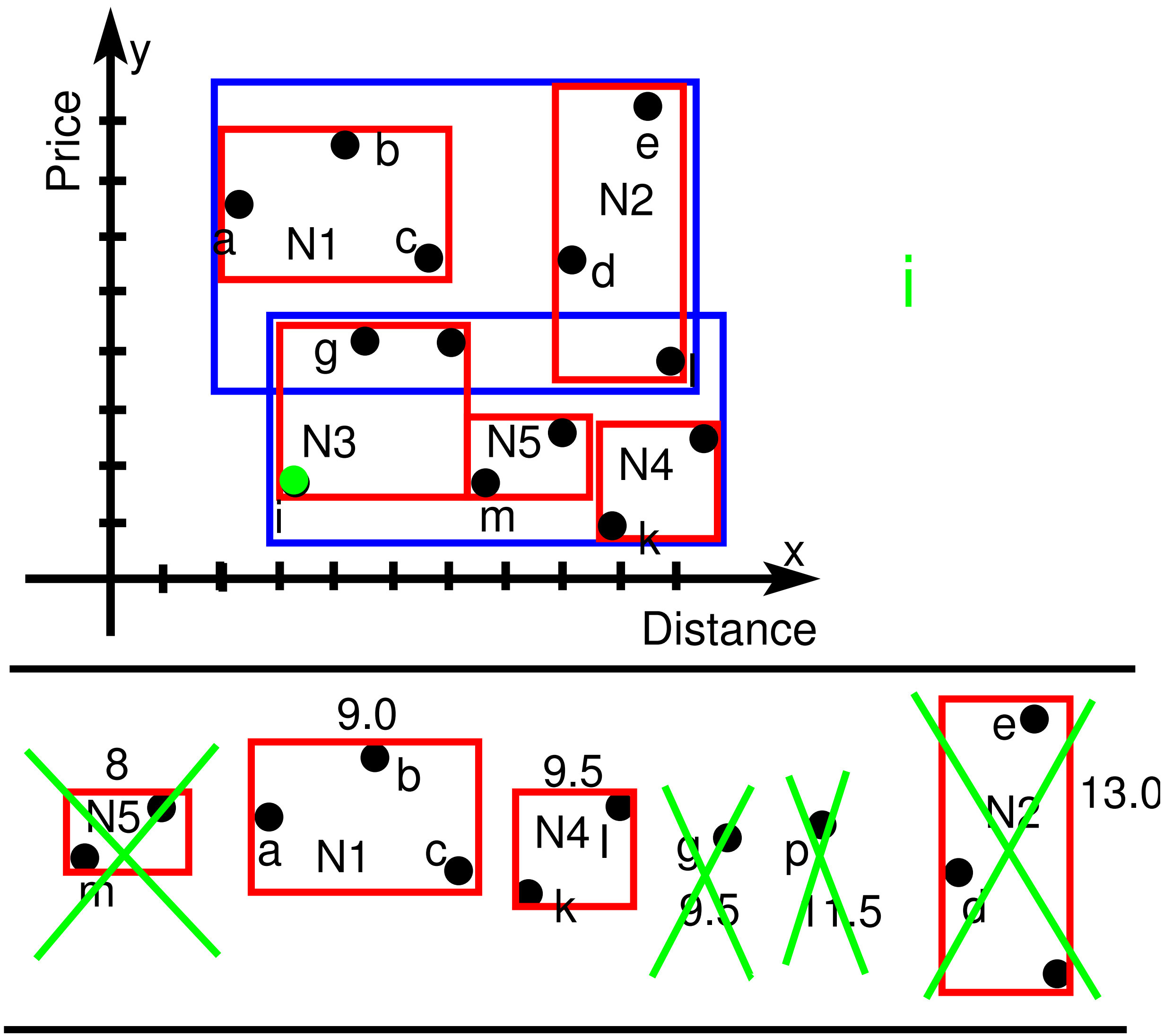
- explode N_3

Branch and Bound Skyline [Papadias, SIGMOD-2003]



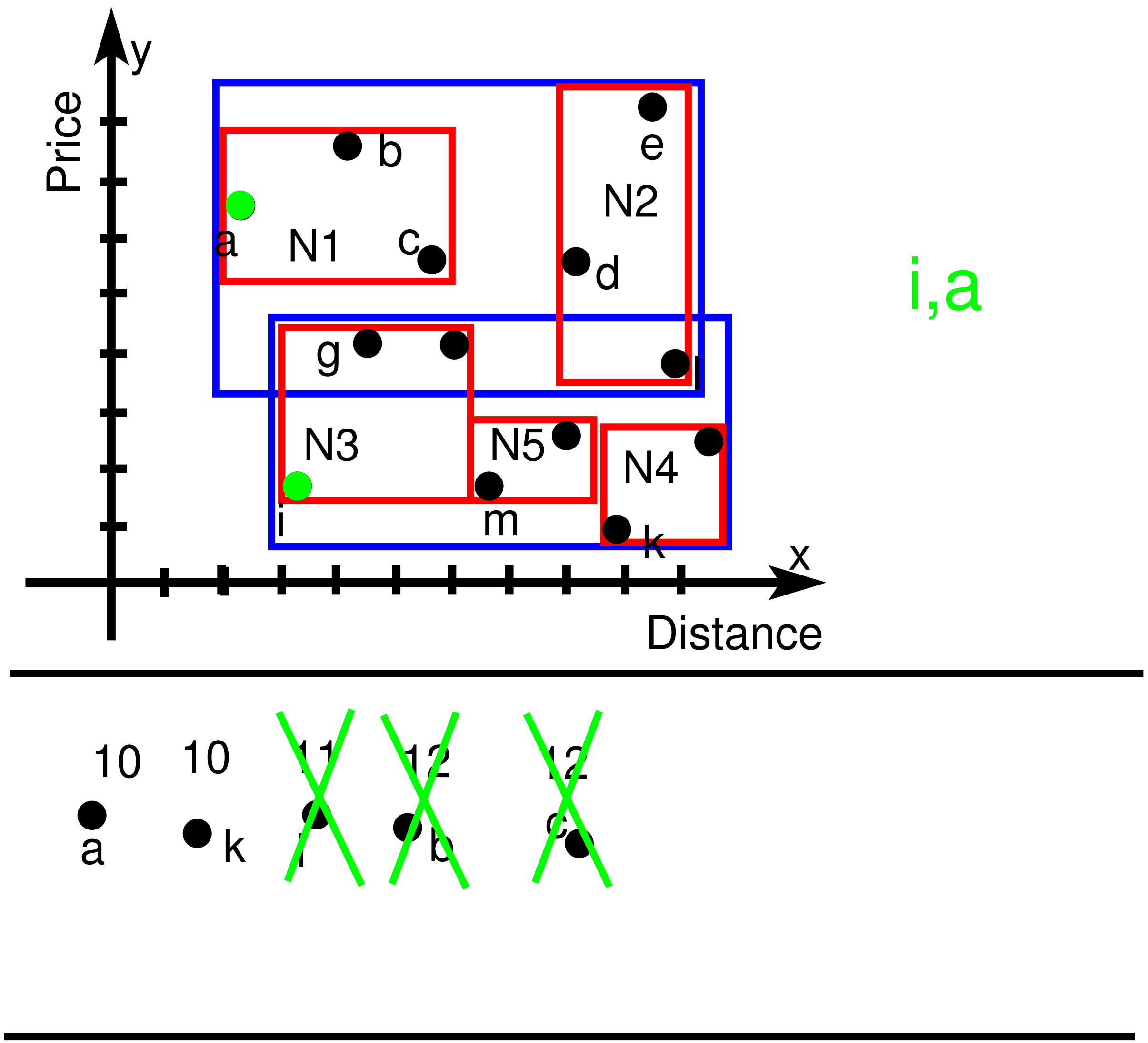
- report i, explode the blue block

Branch and Bound Skyline [Papadias, SIGMOD-2003]



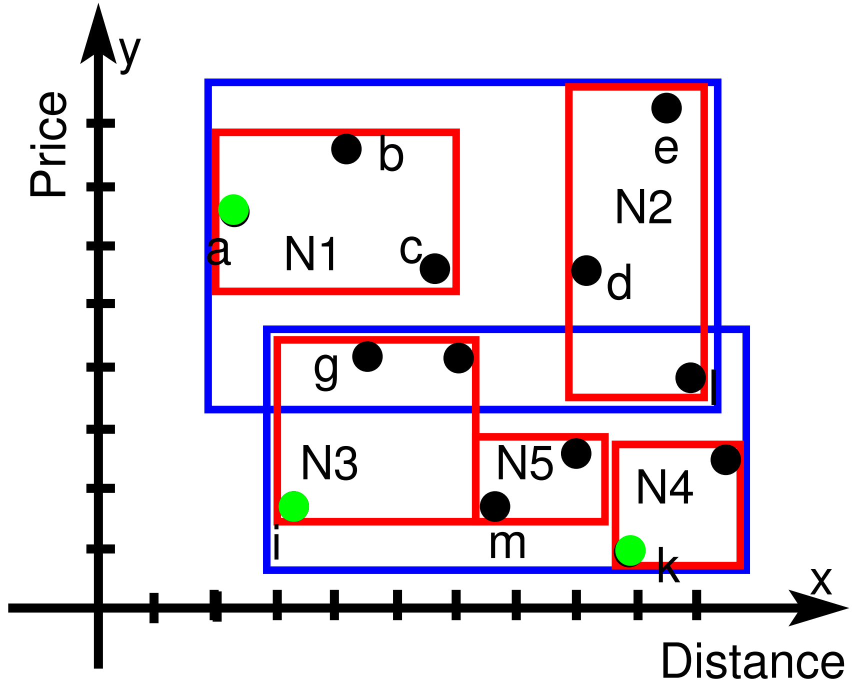
- remove $N5$, g, p, $N2$ as they are all dominated by i, explode $N1$

Branch and Bound Skyline [Papadias, SIGMOD-2003]



- report a. remove i, b, c. Dominated by either i or a.

Branch and Bound Skyline [Papadias, SIGMOD-2003]



i,a,k

- report k

Variations of the skyline queries

- Ranked skyline queries: an alternate *preference function* is used instead of the minimum criterion.
  - The priority queue uses the alternate preference-function to compute *MinDist* to the elements in the queue
- Constrained skyline queries: The skyline query returns skyline points only from the data-space defined by the constraint
  - when inserting objects into the priority queue, prune objects that completely lie outside the constraint region.
- Enumerating queries: For each skyline point in the dataset, find the number of points in the dataset dominated by it.
  - Identify the skyline points; define the spatial bounds for the region where a skyline point dominates.
  - Scan all points in the dataset and check it against the spatial extent for each of the skyline point. The total number of point-region intersection gives the required count for each skyline point.
- K-Dominating queries retrieves the K points that dominate the largest number of points in the dataset.