Caching Dynamic Skyline Queries

D. Sacharidis[1], **P. Bouros**[1], T. Sellis[1,2]

[1]National Technical University of Athens

[2]Institute for Management of Information Systems – R.C. Athena

Outline

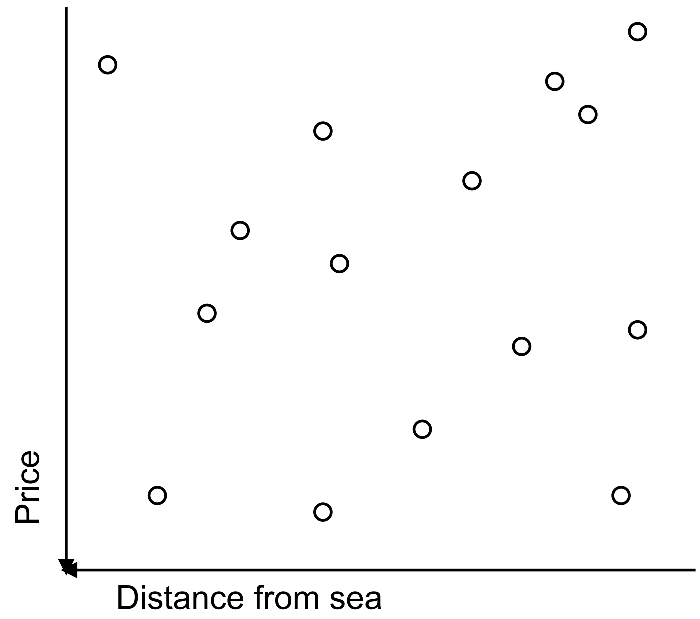
- Introduction
  - Skyline (SL) and dynamic skyline queries (DSL)
- Related work
- Evaluating dynamic skyline queries
  - Computing orthant skylines (OSL)
  - Computing dynamic skyline via caching
    - LRU, LFU, LPP cache replacement policies
- Experimental evaluation
- Conclusions and Future work

Skyline queries (SL)

- Given a dataset of d-dimensional points
  - SL contains points not dominated by others
  - x dominates y iff x as good as y in all dimensions and strictly better in at least one

Skyline queries (SL)

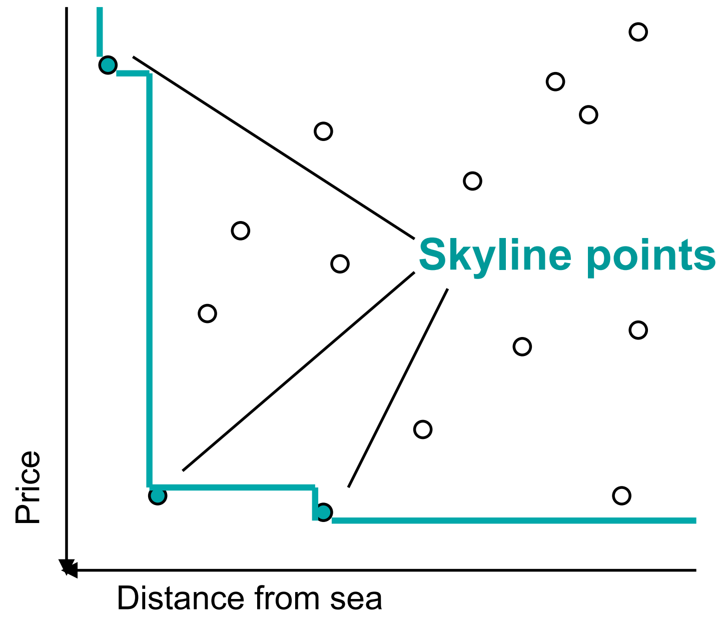
- Given a dataset of d-dimensional points
  - SL contains points not dominated by others
  - x dominates y iff x as good as y in all dimensions and strictly better in at least one



- Example
  - Dataset of hotels
  - Prefer cheap hotels close to the sea

Skyline queries (SL)

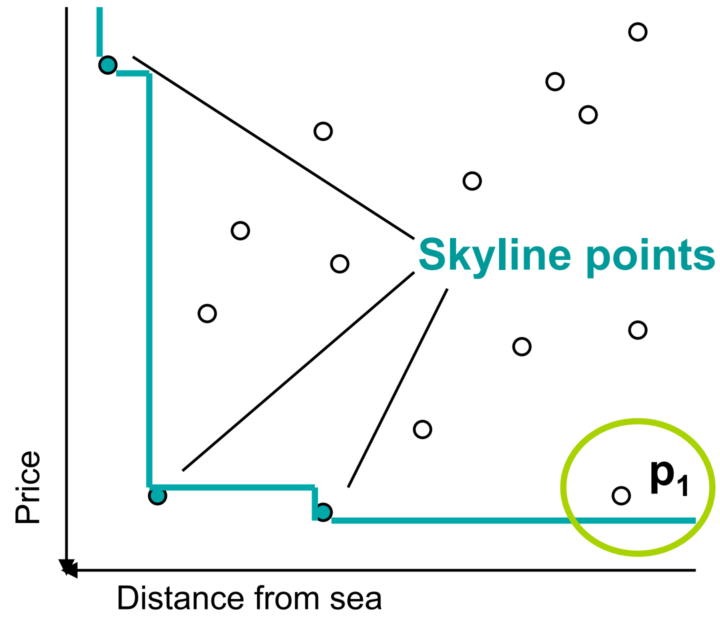
- Given a dataset of d-dimensional points
  - SL contains points not dominated by others
  - x dominates y iff x as good as y in all dimensions and strictly better in at least one



- Example
  - Dataset of hotels
  - Prefer cheap hotels close to the sea

Skyline queries (SL)

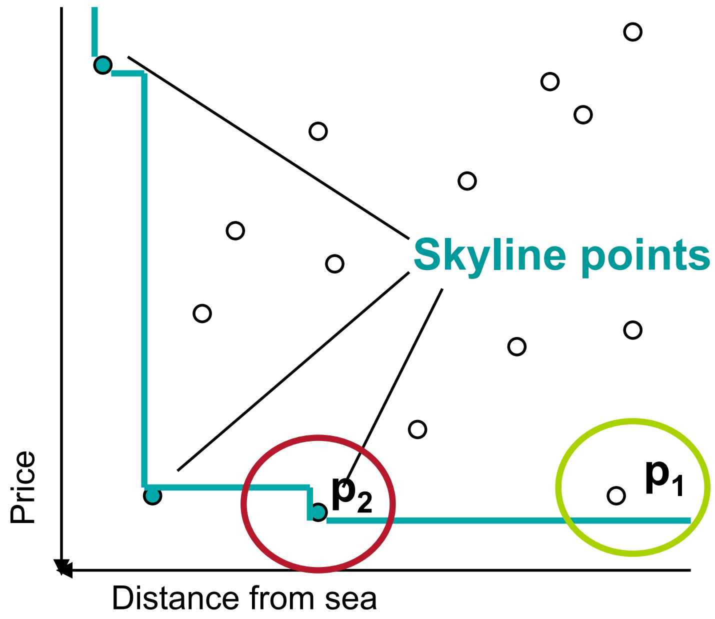
- Given a dataset of d-dimensional points
  - SL contains points not dominated by others
  - x dominates y iff x as good as y in all dimensions and strictly better in at least one



- Example
  - Dataset of hotels
  - Prefer cheap hotels close to the sea

Skyline queries (SL)

- Given a dataset of d-dimensional points
  - SL contains points not dominated by others
  - x dominates y iff x as good as y in all dimensions and strictly better in at least one



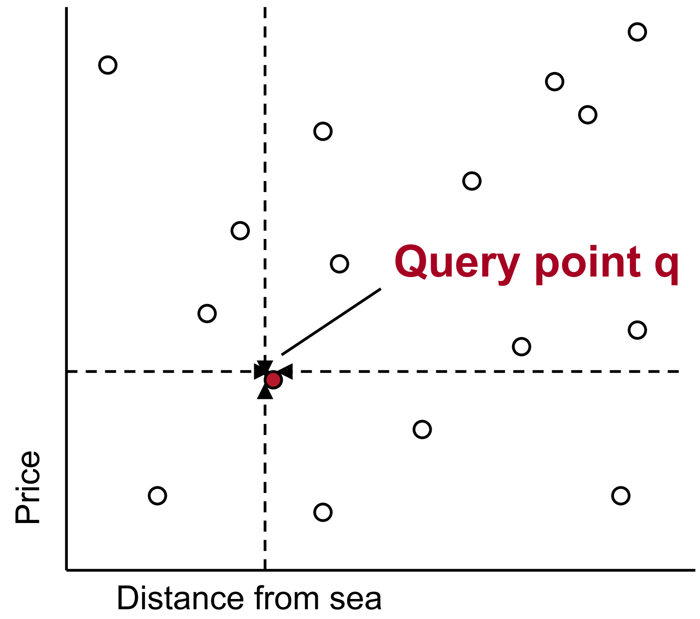
- Example
  - Dataset of hotels
  - Prefer cheap hotels close to the sea

Dynamic skyline queries (DSL)

- Extension of skyline queries
  - Given a query point q
  - DSL contains points not dynamically dominated by others w.r.t q
  - x dynamically dominates y iff x as preferable as y w.r.t. q in all dimensions and strictly more preferable w.r.t. q in at least one
- Can be treated as static SL
  - Transform points w.r.t. q

Dynamic skyline queries (DSL)

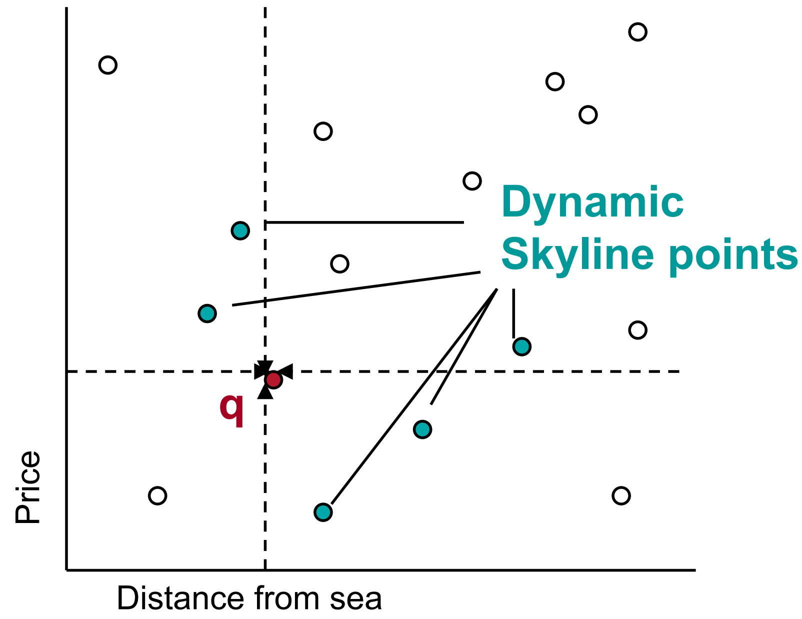
- Extension of skyline queries
  - Given a query point q
  - DSL contains points not dynamically dominated by others w.r.t q
  - x dynamically dominates y iff x as preferable as y w.r.t. q in all dimensions and strictly more preferable w.r.t. q in at least one
- Can be treated as static SL
  - Transform points w.r.t. q



- Example
  - User defines “ideal” hotel q

Dynamic skyline queries (DSL)

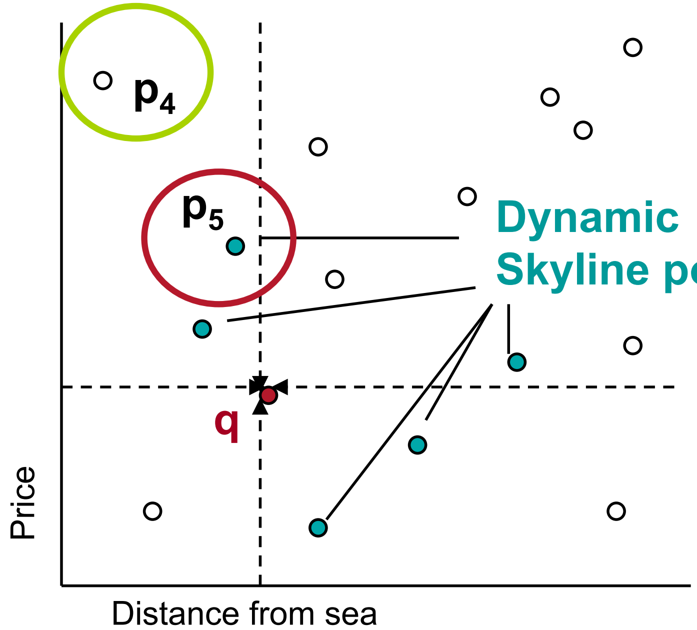
- Extension of skyline queries
  - Given a query point q
  - DSL contains points not dynamically dominated by others w.r.t q
  - x dynamically dominates y iff x as preferable as y w.r.t. q in all dimensions and strictly more preferable w.r.t. q in at least one
- Can be treated as static SL
  - Transform points w.r.t. q



- Example
  - User defines “ideal” hotel q

Dynamic skyline queries (DSL)

- Extension of skyline queries
  - Given a query point q
  - DSL contains points not dynamically dominated by others w.r.t q
  - x dynamically dominates y iff x as preferable as y w.r.t. q in all dimensions and strictly more preferable w.r.t. q in at least one
- Can be treated as static SL
  - Transform points w.r.t. q

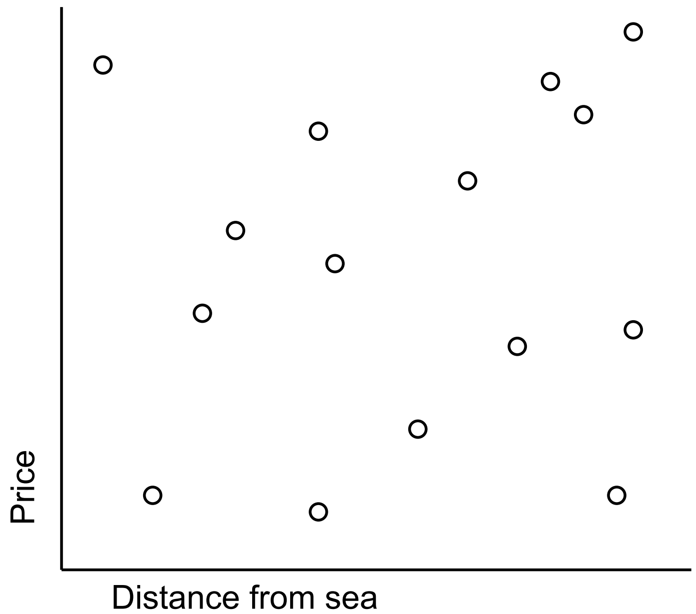


- Example
  - User defines "ideal" hotel q

Intuition (1)

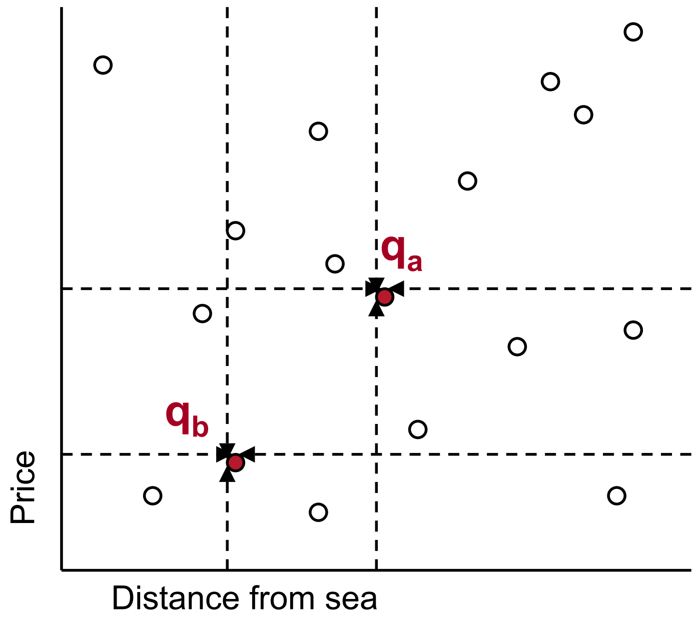
- Traditional SL algorithms need to run anew for each DSL query
- Our idea
  - Exploit results from past queries to reduce processing cost for future DSL queries
  - Cache past queries
  - Decide which queries in cache are useful

Intuition (2)



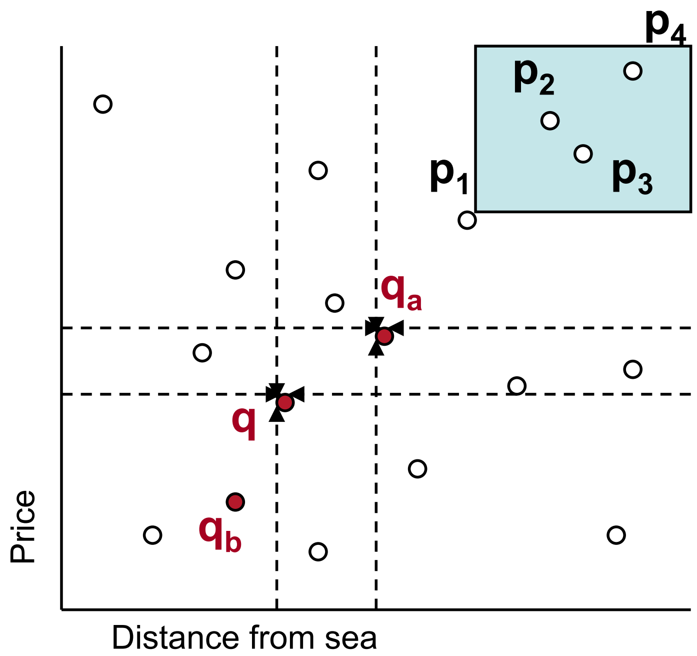
Intuition (2)

- 2 past DSL queries
  - q_a, q_b
- Each query partitions space in 4 quadrants



Intuition (3)

- A new query q arrives
- Consider DSL for q_a
  - p_1 is contained DSL(q_a)
  - p_1 dominates p_2, p_3, p_4
- p_1 lies in upper right quadrant w.r.t. q_a
- q_a lies in upper right quadrant w.r.t. q
- p_1 dominates also p_2, p_3, p_4 w.r.t. to q
  - Exclude p_2, p_3, p_4 from dominance test for DSL(q)



- Shaded area denotes points dominated by p_1

Contribution in brief

- Caching past DSL queries cannot reduce processing cost for future ones
  - We need more information about dominance relationships
- Introduce orthant skylines (OSL) and examine their relationship with DSL
- Extend Bitmap algorithm to compute OSL in parallel with DSL
- Cache OSL to enhance DSL queries evaluation
  - Present 3 cache replacement policies
    - LRU, LFU, LPP
- Experimental evaluation of caching mechanism

Related work

- Non-indexed methods
  - Block-Nested Loops (BnL)
  - Bitmap
  - Multidimensional Divide and Conquer (DnC)
  - Sort First Scan (SFS)
- Index-based methods
  - B-tree
    - sort points according to the lowest valued coordinate
  - R-tree
    - Nearest neighbor based (NN)
    - Branch and bound (BBS)

Related work

- Non-indexed methods
  - Block-Nested Loops (BnL)
  - Bitmap
  - Multidimensional Divide and Conquer (DnC)
  - Sort First Scan (SFS)
- Index-based methods
  - B-tree
    - sort points according to the lowest valued coordinate
  - R-tree
    - Nearest neighbor based (NN)
    - Branch and bound (BBS)

Bitmap

- BnL variant
- Suitable for domains with low cardinality and discrete
- In brief
  - Computes a bitmap representation of the points in the dataset
  - Examines each point separately (dominance test)
    - Checks whether it is contained in the skyline or not
    - Exploits fast bitwise operations OR/AND

Bitmap – Dominance test

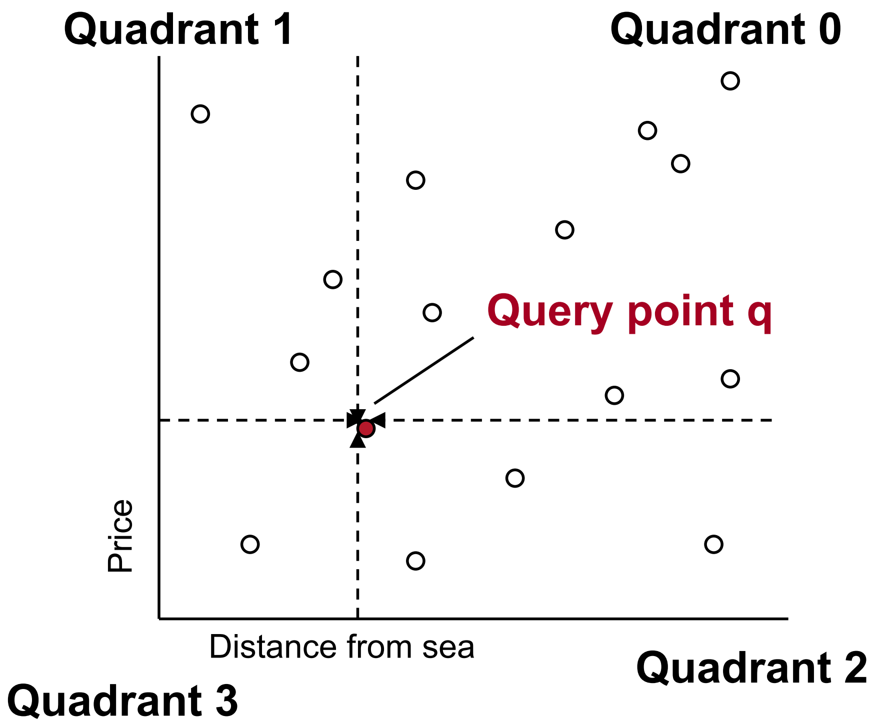
- For each point p
  - Define $A = A^1 \& A^2 \& \ldots \& A^d$
    - Denotes the points as good as p in all dimensions
  - Define $B = B^1 \mid B^2 \mid \ldots \mid B^d$
    - Denotes the points strictly better than p in at least one dimension
  - Dominance test:
    - If $C = A \& B$ has all bits set to 0 then p is in SL

Orthant skyline (OSL)

- OSL provides more information about dominance relationships than DSL
  - Useful for pruning
- Given a dataset of d-dimensional points and a query point q
  - Space partitioned in 2^d orthants
  - o-th orthant skyline (OSL) of q contains points of the o-th orthant not dynamically dominated by others inside orthant o w.r.t q

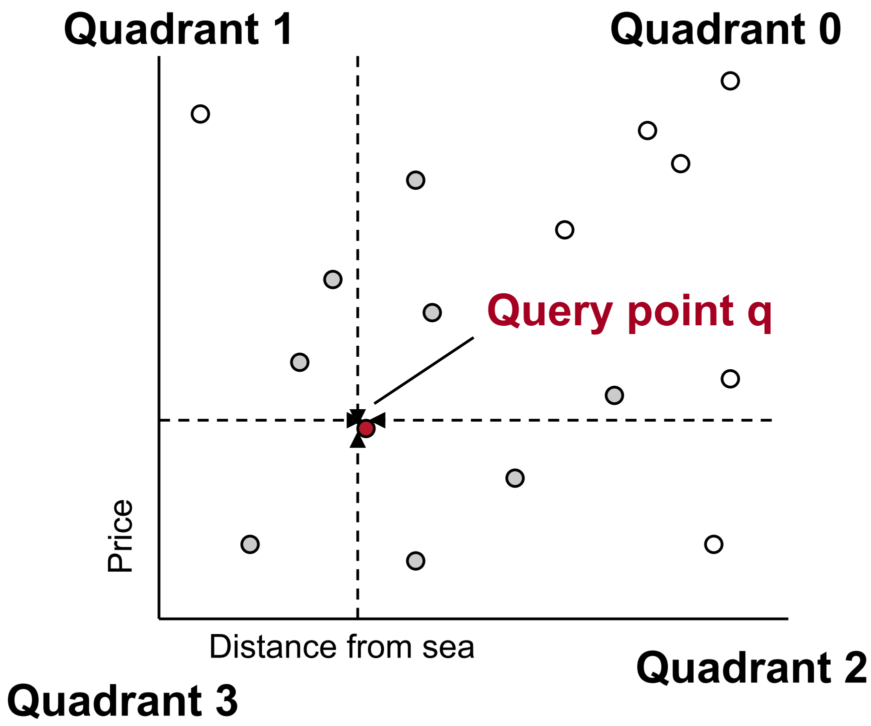
Orthant skyline (OSL)

- OSL provides more information about dominance relationships than DSL
  - Useful for pruning
- Given a dataset of d-dimensional points and a query point q
  - Space partitioned in 2^d orthants
  - o-th orthant skyline (OSL) of q contains points of the o-th orthant not dynamically dominated by others inside orthant o w.r.t q



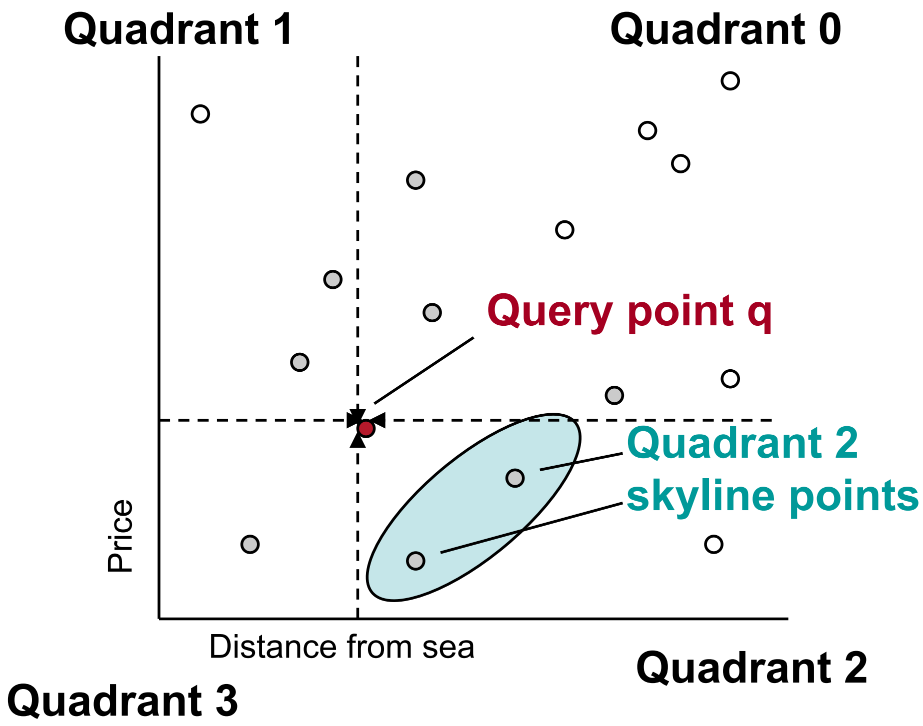
Orthant skyline (OSL)

- OSL provides more information about dominance relationships than DSL
  - Useful for pruning
- Given a dataset of d-dimensional points and a query point q
  - Space partitioned in 2^d orthants
  - o-th orthant skyline (OSL) of q contains points of the o-th orthant not dynamically dominated by others inside orthant o w.r.t q

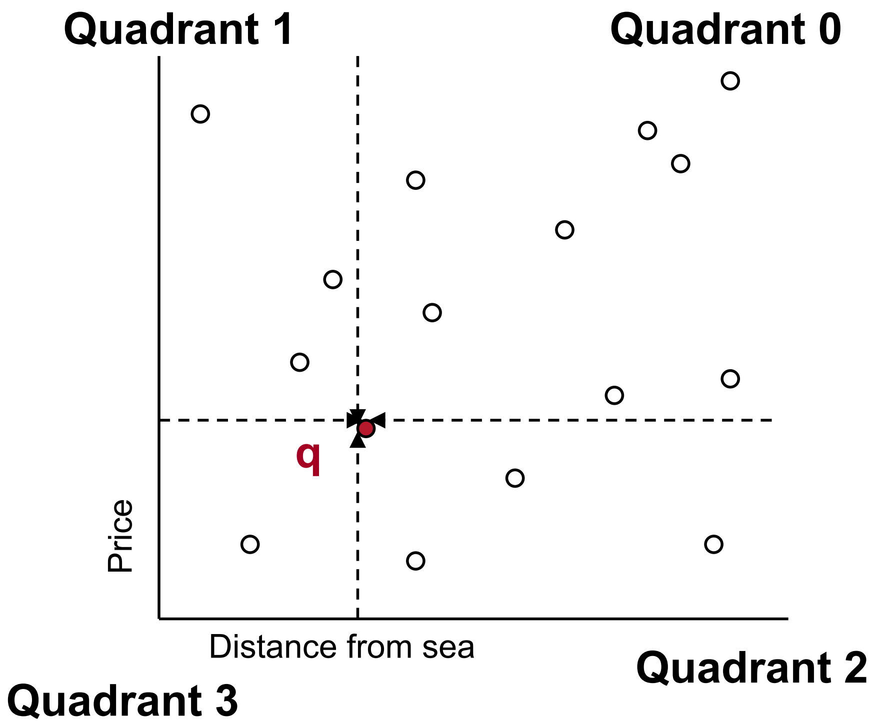


Orthant skyline (OSL)

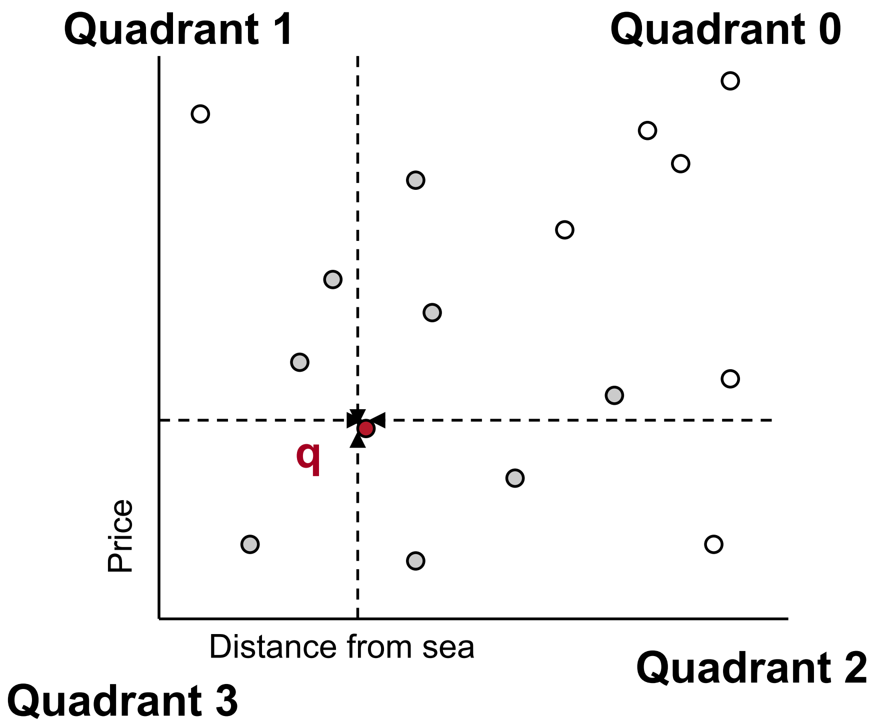
- OSL provides more information about dominance relationships than DSL
  - Useful for pruning
- Given a dataset of d-dimensional points and a query point q
  - Space partitioned in 2^d orthants
  - o-th orthant skyline (OSL) of q contains points of the o-th orthant not dynamically dominated by others inside orthant o w.r.t q



OSL and DSL relationship

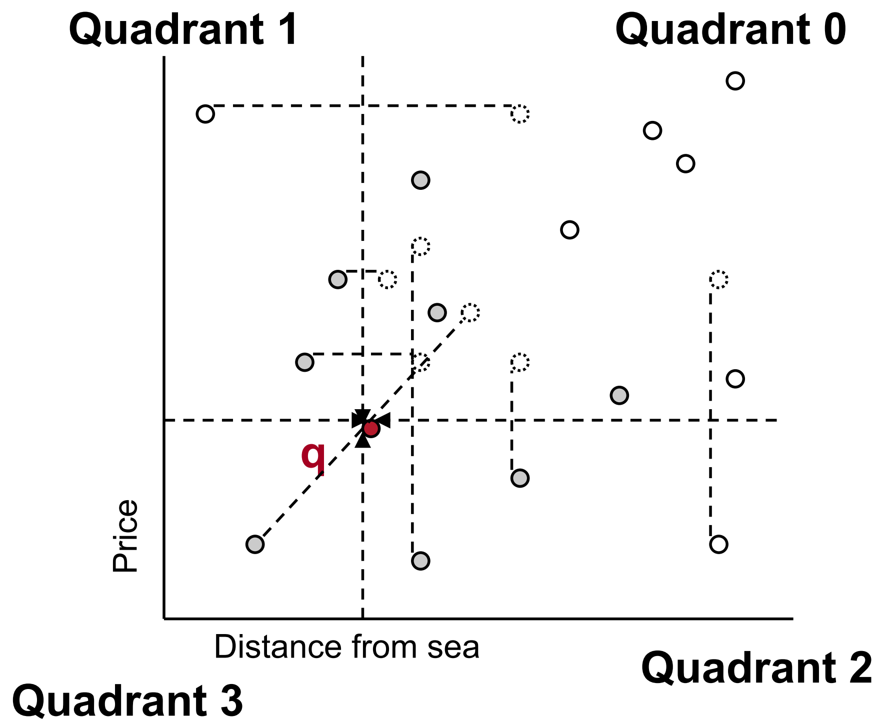


OSL and DSL relationship



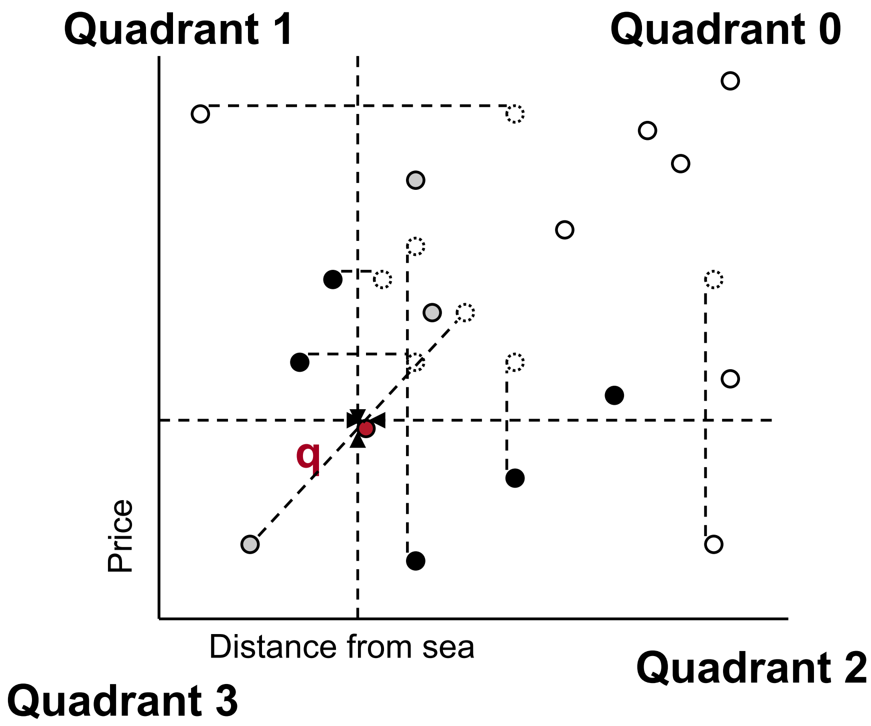
OSL and DSL relationship

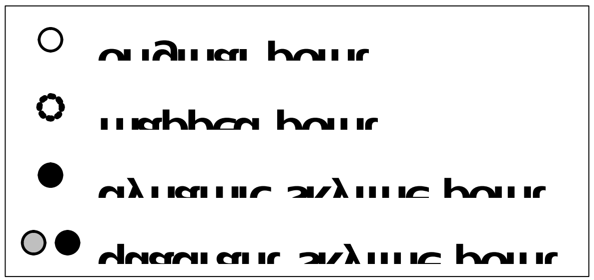
- Map points from quadrants 1,2,3 to points inside quadrant 0



OSL and DSL relationship

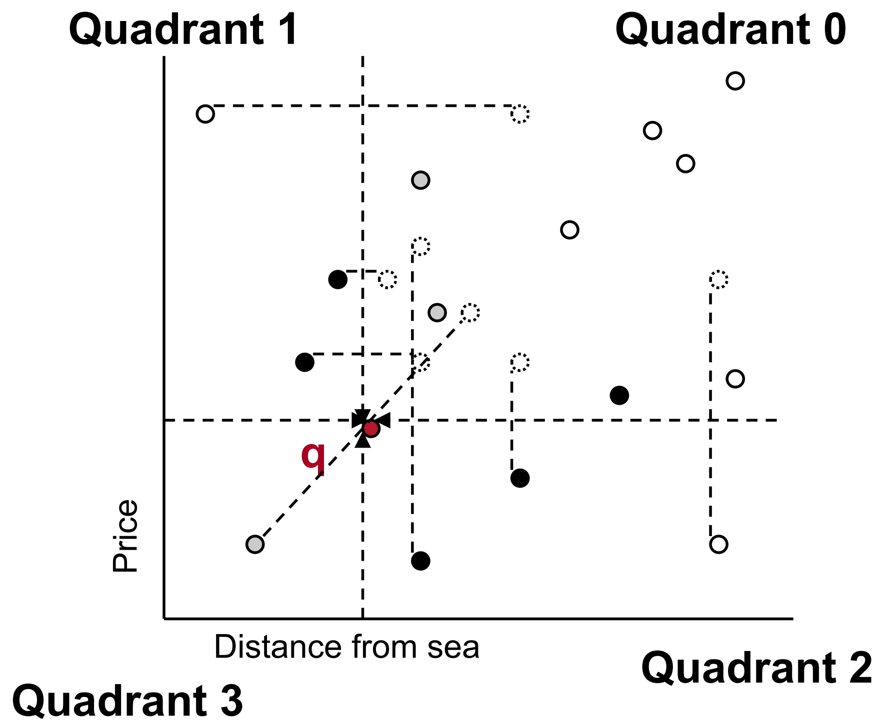
- Map points from quadrants 1,2,3 to points inside quadrant 0
- Compute DSL w.r.t. q

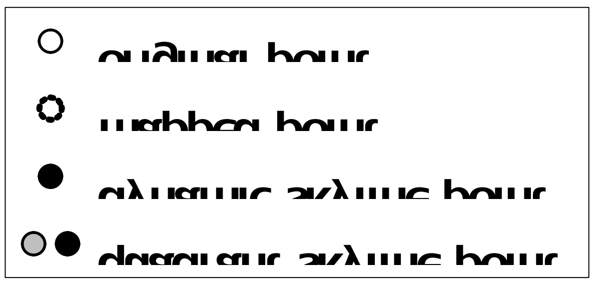




OSL and DSL relationship

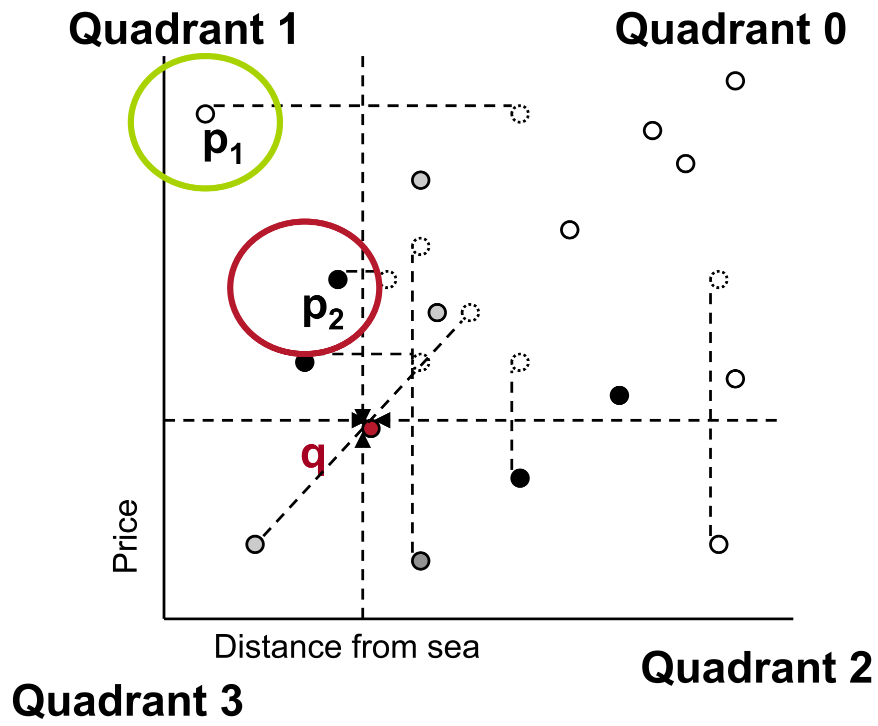
- Map points from quadrants 1,2,3 to points inside quadrant 0
- Compute DSL w.r.t. q
- Union of all OSLs is superset of DSL w.r.t. to q

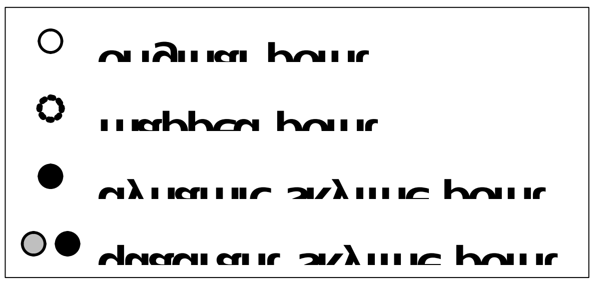




OSL and DSL relationship

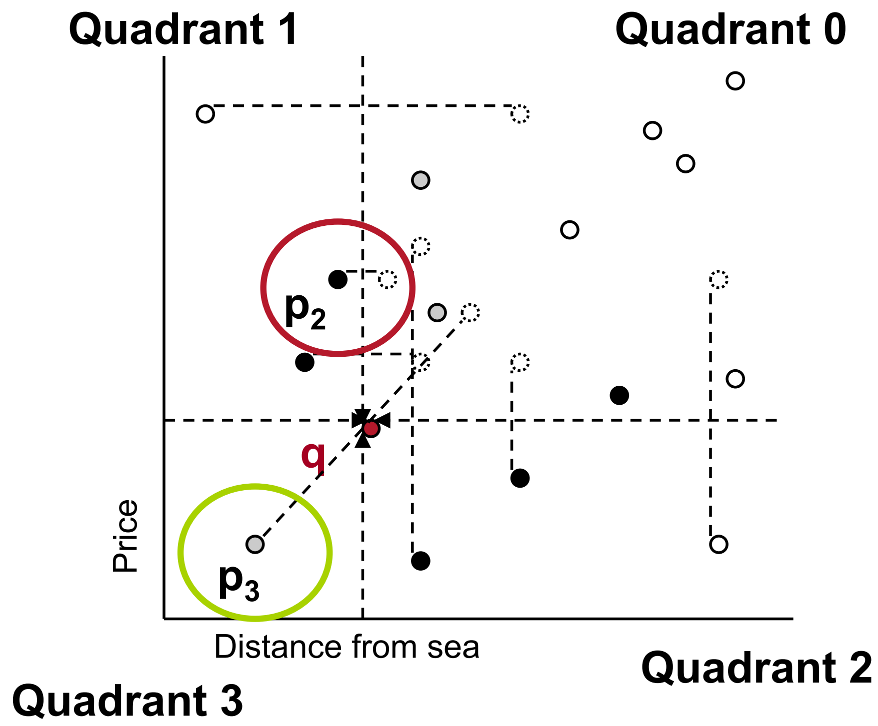
- Map points from quadrants 1,2,3 to points inside quadrant 0
- Compute DSL w.r.t. q
- Union of all OSLs is superset of DSL w.r.t. to q

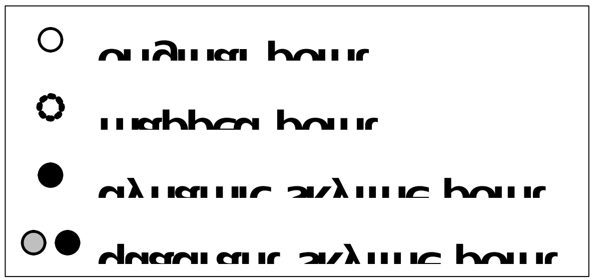




OSL and DSL relationship

- Map points from quadrants 1,2,3 to points inside quadrant 0
- Compute DSL w.r.t. q
- Union of all OSLs is superset of DSL w.r.t. to q





Computing orthant skylines

- Algorithm DBM
  - Extends Bitmap to compute DSL and OSLs at the same time
- Method:
  - Compute bitmap representation
    - Transform each point coordinates w.r.t. to query q
  - Dominance test, point p, orthant o
    - p not in OSL_o and not in DSL
    - p not in DSL, but in OSL_o
    - p in DSL and in OSL_o

Dynamic skylines Via Caching

- Cache OSLs instead of DSLs
  - Query cache contains (query point q_j, OSLs)
  - OSLs encode by bitmaps
- Algorithm cDBM
  - OSL contains information about dominance test inside orthant
  - Discard points inside orthants from dominance tests
- Method:
  - Compute bitmap representation
  - For each point p consider its position (orthant) w.r.t. to cache queries q_j
  - If p in the same orthant o w.r.t q_j as q_j w.r.t. q and p not in $OSL_o(q_j)$ then exclude it from $OSL_o(q)$, $DSL(q)$

Cache Replacement Policies

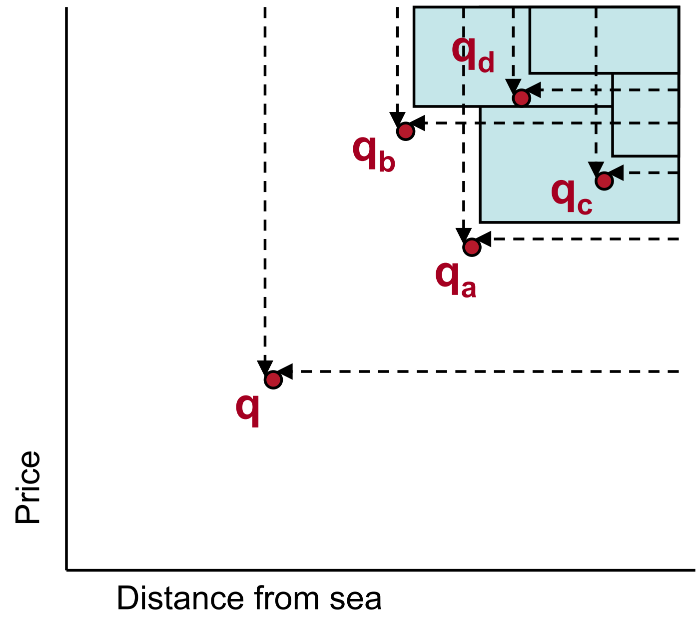
- General idea
  - Limited cache space
  - Identify least useful query point in cache
  - Replace it with new one

Usage-based policies

- Only a few queries in cache are useful
- Log cache query usage

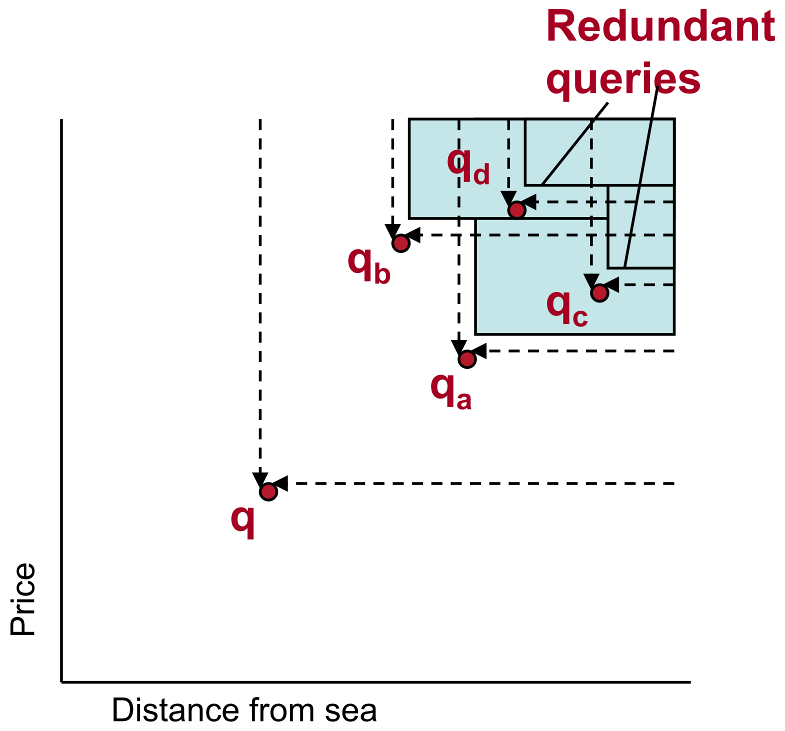
Usage-based policies

- Only a few queries in cache are useful
- Log cache query usage



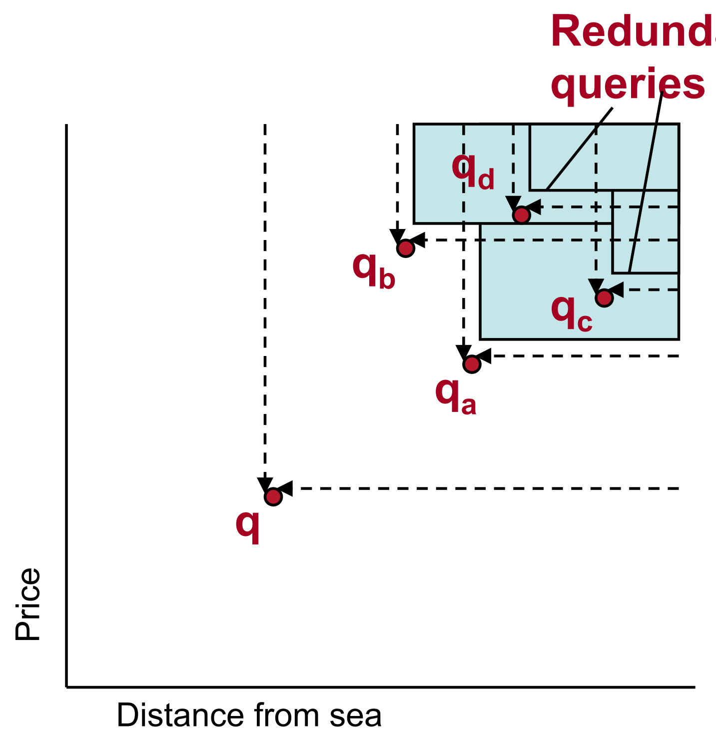
Usage-based policies

- Only a few queries in cache are useful
- Log cache query usage



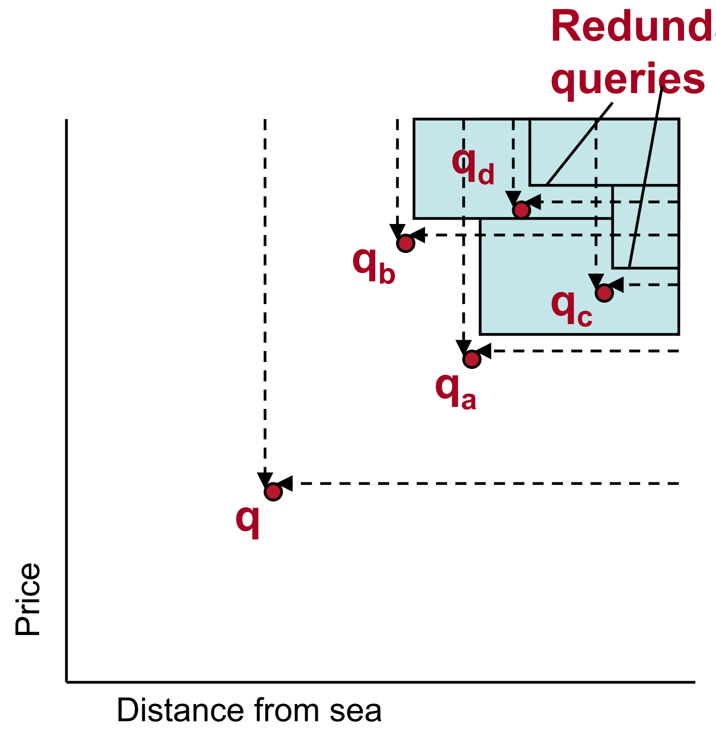
Usage-based policies

- Only a few queries in cache are useful
- Log cache query usage
- Given a new query q
  - Consider as input the query points in cache Q
  - Only query points in OSL of Q w.r.t. q are useful



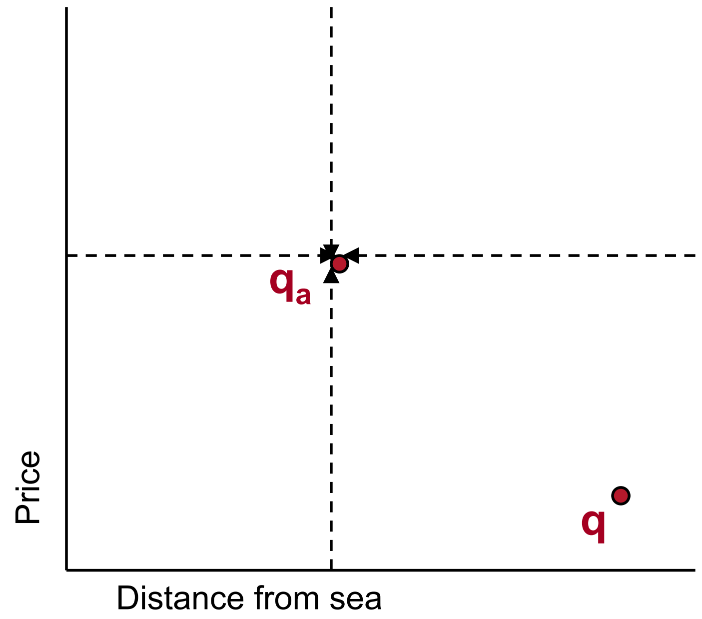
Usage-based policies

- Only a few queries in cache are useful
- Log cache query usage
- Given a new query q
  - Consider as input the query points in cache Q
  - Only query points in OSL of Q w.r.t. q are useful
  - Update cache - remove:
    - Least Recently Used (LRU) query point
    - Least Frequently Used (LFU) query point



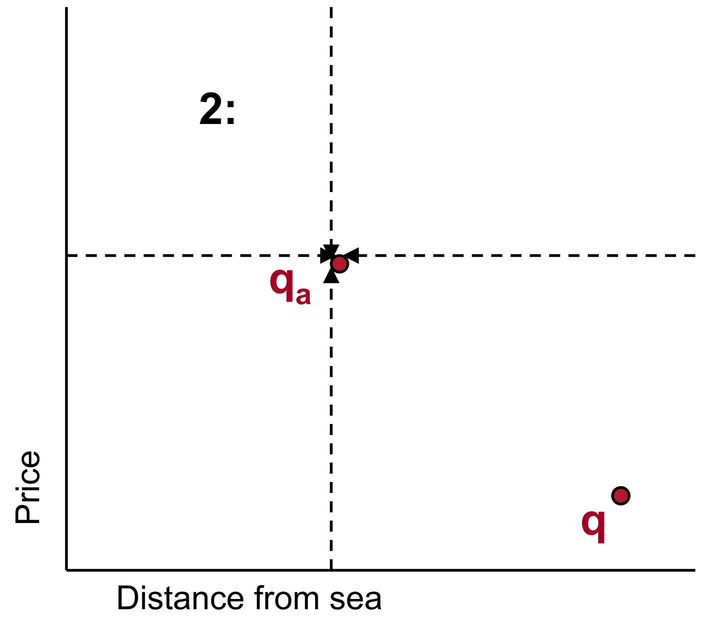
Pruning power-based policy

- Usage-based policies do not indicate usefulness
- Useful cached query
  - Great pruning power
    - Probability that a query can prune points of dataset from DSL computation



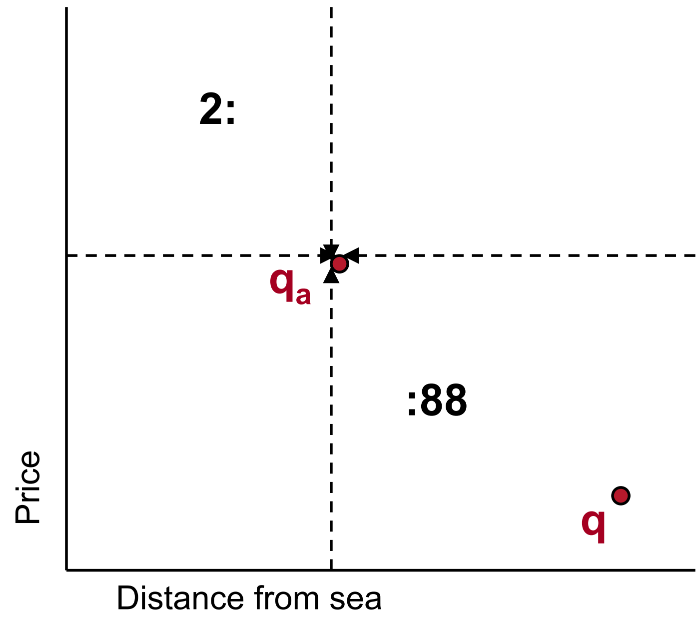
Pruning power-based policy

- Usage-based policies do not indicate usefulness
- Useful cached query
  - Great pruning power
    - Probability that a query can prune points of dataset from DSL computation
  - Depends on
    - Points dominated by query in an orthant j



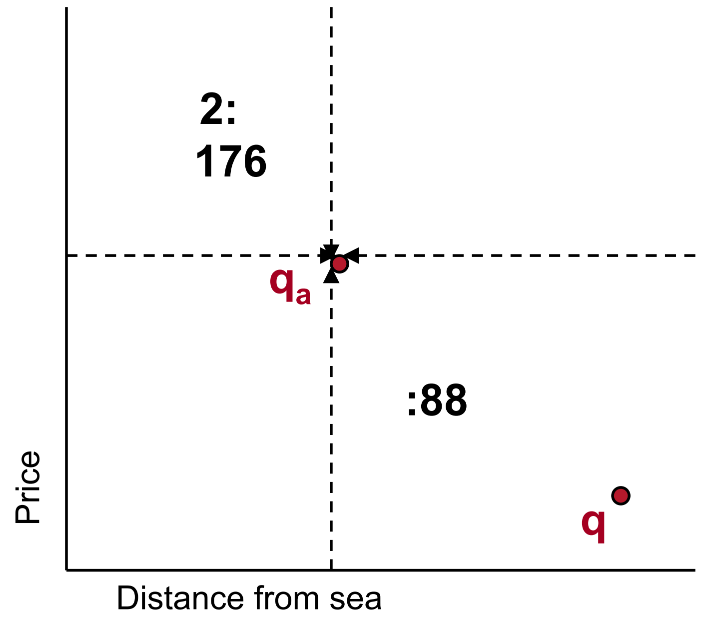
Pruning power-based policy

- Usage-based policies do not indicate usefulness
- Useful cached query
  - Great pruning power
    - Probability that a query can prune points of dataset from DSL computation
  - Depends on
    - Points dominated by query in an orthant j
    - Points contained in the antisymetric orthant of j



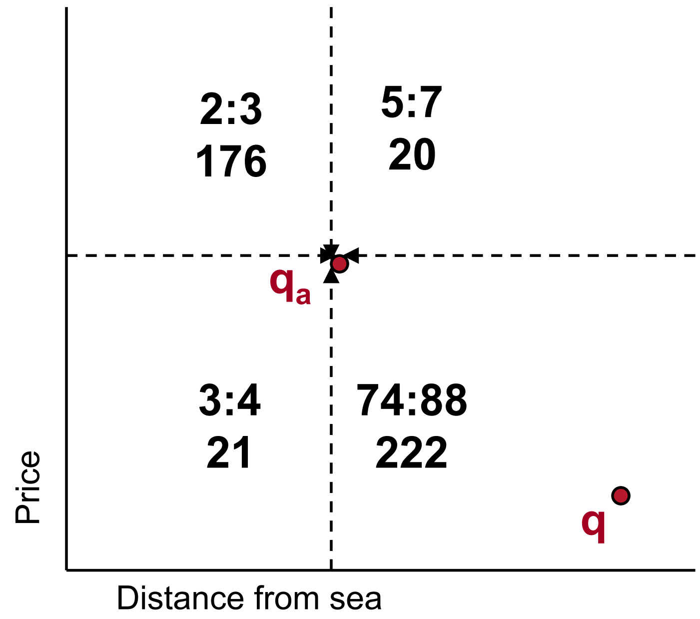
Pruning power-based policy

- Usage-based policies do not indicate usefulness
- Useful cached query
  - Great pruning power
    - Probability that a query can prune points of dataset from DSL computation
  - Depends on
    - Points dominated by query in an orthant j
    - Points contained in the antisymetric orthant of j



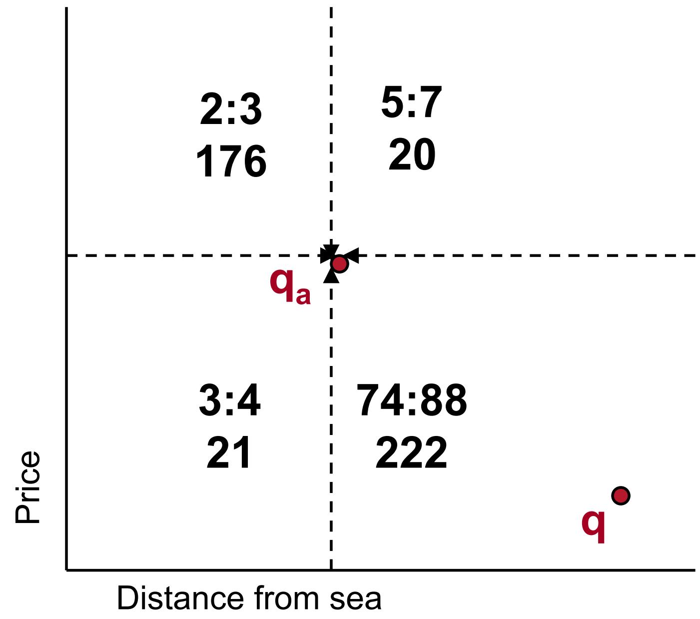
Pruning power-based policy

- Usage-based policies do not indicate usefulness
- Useful cached query
  - Great pruning power
    - Probability that a query can prune points of dataset from DSL computation
  - Depends on
    - Points dominated by query in an orthant j
    - Points contained in the antisymetric orthant of j



Pruning power-based policy

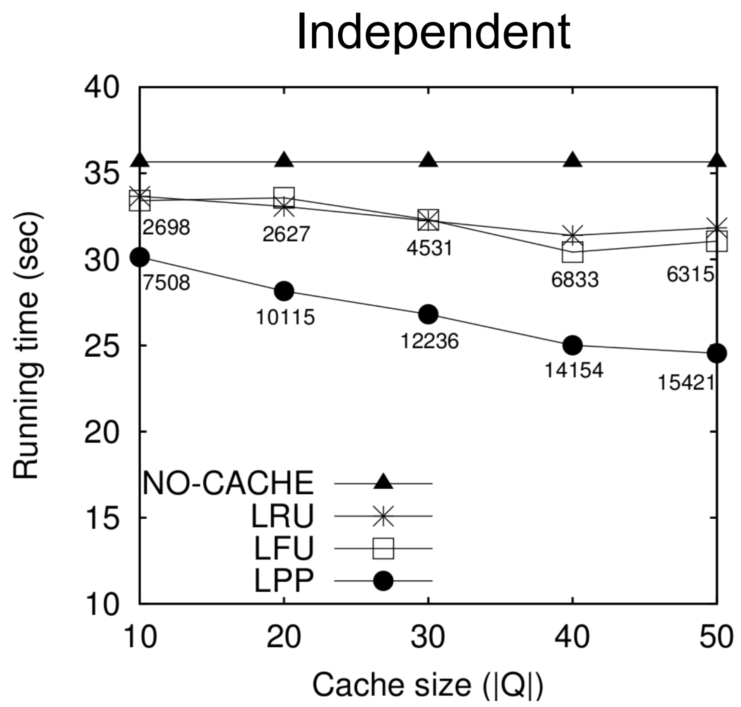
- Usage-based policies do not indicate usefulness
- Useful cached query
  - Great pruning power
    - Probability that a query can prune points of dataset from DSL computation
  - Depends on
    - Points dominated by query in an orthant j
    - Points contained in the antisymetric orthant of j
- Update cache – remove
  - Query point with less pruning power (LPP)

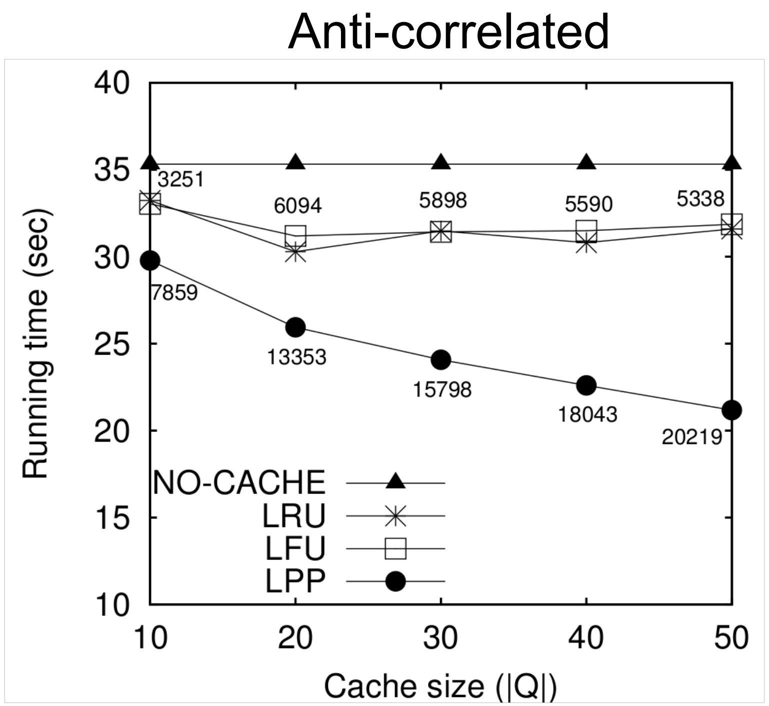


Experimental Evaluation

- Synthetic datasets
  - Distribution types
    - Independent, correlated, anti-correlated
  - Number of points N
    - 10k, 20k, 50k, 100k,
  - Dimensionality
    - d = {2,3,4,5,6}
  - Domain size for dimension
    - |D| = {10,20,50}
- Compare
  - Bitmap (NO-CACHE)
  - cDBM with LFU,LRU,LPP cache replacement policies
  - Query cache
    - |Q| = {10,20,30,40,50} past query points
    - Cache size is |Q|*N bits uncompressed

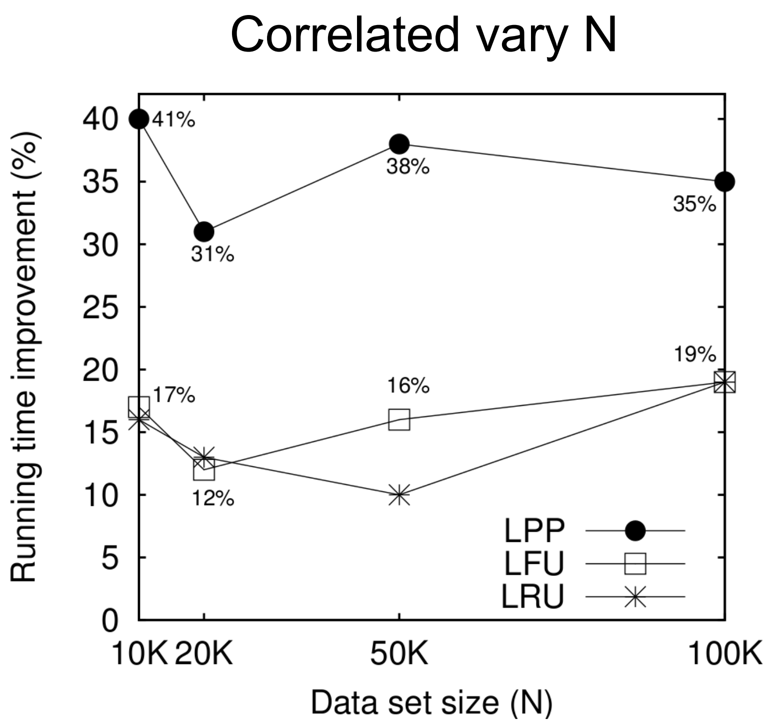
Varying query cache size

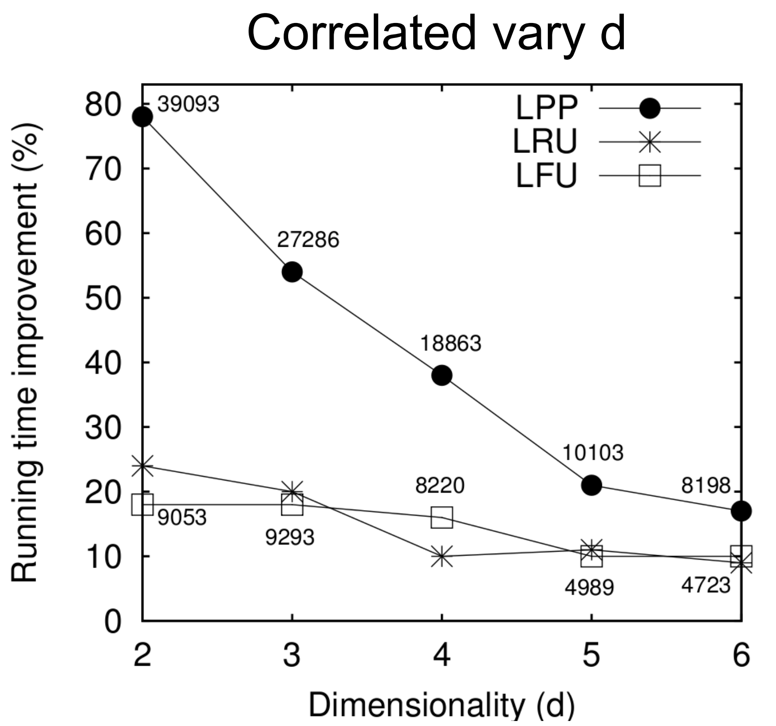




- Dataset: N = 50k points, with d = 4 dimensions of |D| = 20 domain size
- LFU,LRU cache queries not representative for future ones
- LPP caches queries with great pruning power

Effect of distribution parameters





- Relative improvement in running time over NO-CACHE
- Vary number of points N
  - d = 4 dimensions of |D| = 20 domain size
- Vary number of dimensions d
  - N = 50k, |D| = 20

Conclusions and Future work

- Conclusions
  - Introduced orthant skylines (OSLs) and discussed its relationship with DSL
  - Extended Bitmap to compute OSLs and DSL at the same time (DBM algorithm)
  - Proposed caching mechanism of OSLs to reduce cost for future DSL queries
    - LRU, LFU, LPP cache replacement policies
  - Experimentally verified the efficiency of caching mechanism
- Future work
  - Apply caching mechanism to index-based methods
  - Further increase pruning power of cached queries

Questions ?