Dynamic Processing of Dominating Queries with Performance Guarantees

Andreas Kosmatopoulos, Apostolos N. Papadopoulos, and Kostas Tsichlas

Aristotle University of Thessaloniki

{*akosmato,papadopo,tsichlas*}*@csd.auth.gr*

26 March 2014

Introduction

- *Preference-based queries* select the most interesting objects of a given dataset

Introduction

- *Preference-based queries* select the most interesting objects of a given dataset
- Data objects are characterized by (usually) contradictory attributes
  - Price and quality
  - Size and operation speed (in data structures)
- Selecting a suitable result becomes a challenging task

Introduction

- *Preference-based queries* select the most interesting objects of a given dataset

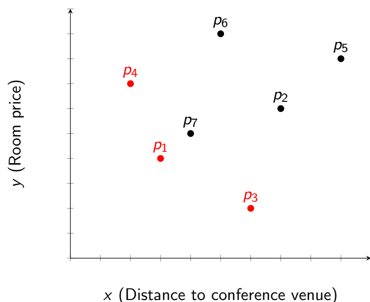
- Data objects are characterized by (usually) contradictory attributes
  - Price and quality
  - Size and operation speed (in data structures)
- Selecting a suitable result becomes a challenging task

- Three main types of preference-based queries
  - Skyline queries
  - Top-k queries
  - Top-k dominating queries

Skyline Query

Dominance

A point p dominates another point q ($p \prec q$) iff p is as good as q in all dimensions and it is strictly better than q in at least one of the dimensions

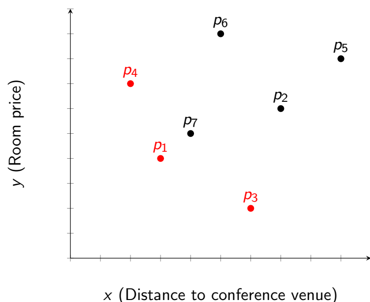


Skyline Query

Dominance

A point p dominates another point q ($p \prec q$) iff p is as good as q in all dimensions and it is strictly better than q in at least one of the dimensions

- A *skyline query* returns all the points that are not dominated by any other point

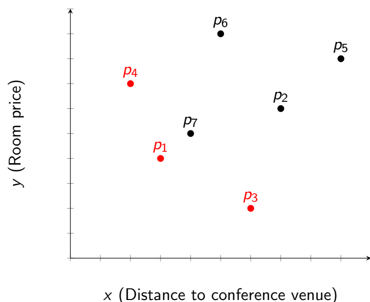


Skyline Query

Dominance

A point p dominates another point q ($p \prec q$) iff p is as good as q in all dimensions and it is strictly better than q in at least one of the dimensions

- A *skyline query* returns all the points that are not dominated by any other point
- (+) The skyline set is invariant in dimension scaling

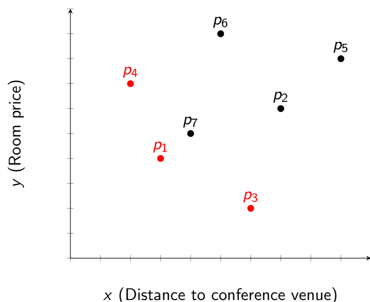


Skyline Query

Dominance

A point p dominates another point q ($p \prec q$) iff p is as good as q in all dimensions and it is strictly better than q in at least one of the dimensions

- A *skyline query* returns all the points that are not dominated by any other point
- (+) The skyline set is invariant in dimension scaling
- (+) Does not require a user-defined scoring function

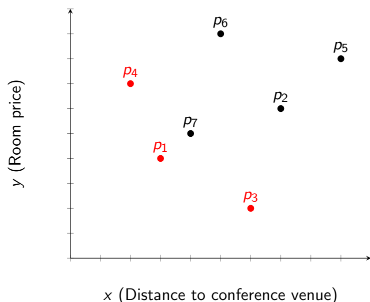


Skyline Query

Dominance

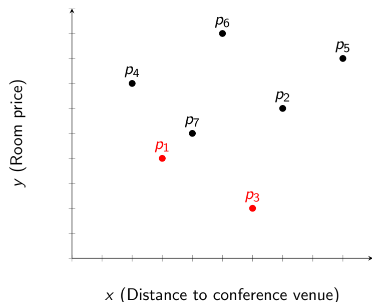
A point p dominates another point q ($p \prec q$) iff p is as good as q in all dimensions and it is strictly better than q in at least one of the dimensions

- A *skyline query* returns all the points that are not dominated by any other point
- (+) The skyline set is invariant in dimension scaling
- (+) Does not require a user-defined scoring function
- (-) The output size is not controlled



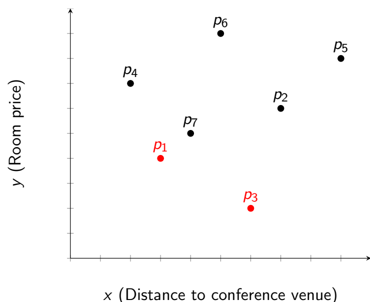
Top-k Query

- A *top-k query* returns exactly k points which are the best according to a user-defined scoring function
  - For the scoring function $f(p) = p.x + p.y$ a top-2 query returns p_1 and p_3



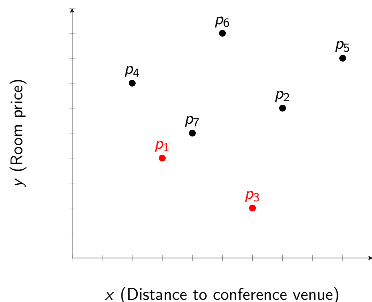
Top-k Query

- A *top-k query* returns exactly k points which are the best according to a user-defined scoring function
  - For the scoring function $f(p) = p.x + p.y$ a top-2 query returns p_1 and p_3
- (+) Output size is controlled through the parameter k



Top-k Query

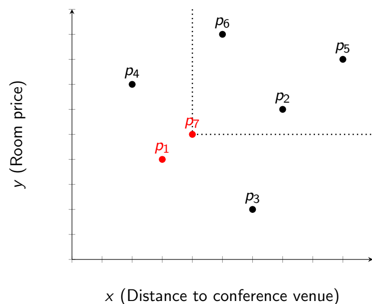
- A *top-k query* returns exactly k points which are the best according to a user-defined scoring function
  - For the scoring function $f(p) = p.x + p.y$ a top-2 query returns p_1 and p_3
- (+) Output size is controlled through the parameter k
- (-) Requires a user-defined scoring function



Top-k Dominating Query

Dominance Score

The dominance score of a point p is the number of points dominated by p

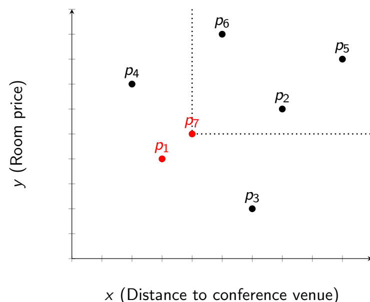


Top-k Dominating Query

Dominance Score

The dominance score of a point p is the number of points dominated by p

- A *top-k dominating query* returns the k points with the highest dominance score
  - A top-2 dominating query returns p_1 and p_7

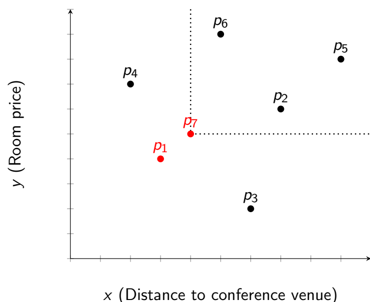


Top-k Dominating Query

Dominance Score

The dominance score of a point p is the number of points dominated by p

- A *top-k dominating query* returns the k points with the highest dominance score
  - A top-2 dominating query returns p_1 and p_7
- Combines the merits of skyline and top-k queries
- (+) Does not depend on the scaling of the dimensions

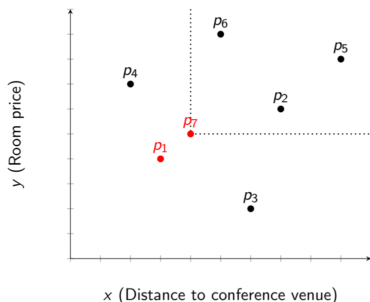


Top-k Dominating Query

Dominance Score

The dominance score of a point p is the number of points dominated by p

- A *top-k dominating query* returns the k points with the highest dominance score
  - A top-2 dominating query returns p_1 and p_7
- Combines the merits of skyline and top-k queries
- (+) Does not depend on the scaling of the dimensions
- (+) Intuitive and definite scoring function

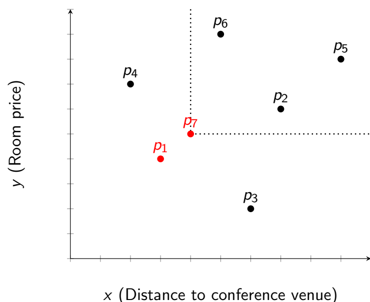


Top-k Dominating Query

Dominance Score

The dominance score of a point p is the number of points dominated by p

- A *top-k dominating query* returns the k points with the highest dominance score
  - A top-2 dominating query returns p_1 and p_7
- Combines the merits of skyline and top-k queries
- (+) Does not depend on the scaling of the dimensions
- (+) Intuitive and definite scoring function
- (+) Controlled output size



Related Work

- Papadias et al. [ToDS'05] first proposed the d-dimensional top-k dominating query
  - The top-1 dominating point is contained in the dataset's skyline points
  - They provided a solution based on the iterative computation of a dataset's skyline points
- Yiu and Mamoulis [VLDB'07, JVLDB'09] recommended using aggregate R-trees (aR-trees)
  - The algorithms they developed proved experimentally to be quite fast

Related Work

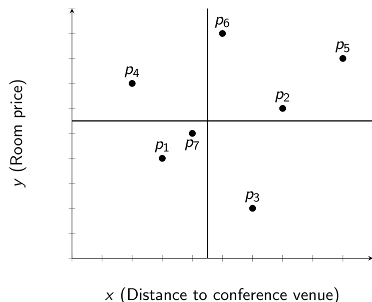
- Papadias et al. [ToDS'05] first proposed the d-dimensional top-k dominating query
  - The top-1 dominating point is contained in the dataset's skyline points
  - They provided a solution based on the iterative computation of a dataset's skyline points
- Yiu and Mamoulis [VLDB'07, JVLDB'09] recommended using aggregate R-trees (aR-trees)
  - The algorithms they developed proved experimentally to be quite fast
- In both methods:
  - k is arbitrary
  - Update operations can be applied with a linear time cost
  - The top-k dominating query has to be re-evaluated after an update

Related Work

- Papadias et al. [ToDS'05] first proposed the d-dimensional top-k dominating query
  - The top-1 dominating point is contained in the dataset's skyline points
  - They provided a solution based on the iterative computation of a dataset's skyline points
- Yiu and Mamoulis [VLDB'07, JVLDB'09] recommended using aggregate R-trees (aR-trees)
  - The algorithms they developed proved experimentally to be quite fast
- In both methods:
  - k is arbitrary
  - Update operations can be applied with a linear time cost
  - The top-k dominating query has to be re-evaluated after an update
- Top-k dominating queries have been studied in uncertain databases, data streams and spatial objects

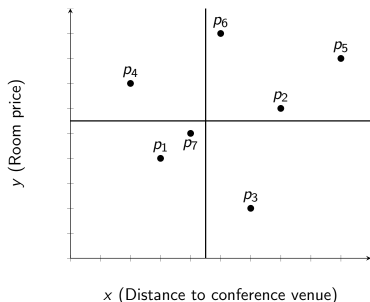
The Problem and Challenges

- Top-k dominating points under a dataset that supports insertions (and deletions)



The Problem and Challenges

- Top-k dominating points under a dataset that supports insertions (and deletions)
- Previous algorithms do not offer asymptotic guarantees
- In contrast to skyline and top-k queries, the top-k dominating query is a *non-decomposable query*
  - Standard divide and conquer techniques may not apply



Contribution

Algorithm	Space	Preprocessing Cost (worst-case)	Query Cost (worst-case)	Update Cost
SD/k-list	$O(n)$	$O(n \log n)$	$O(k)$	$O(\log^2 n + k^2 \log n)$ w.c.
SD/1-list	$O(n)$	$O(n \log n)$	$O(k \log n)$	$O(\log^2 n + k \log n)$ w.c.
FD/k-list	$O(n)$	$O(n \log n)$	$O(k)$	$O((k + \sqrt{n})k \log n)$ am.
FD/1-list	$O(n)$	$O(n \log n)$	$O(k \log n)$	$O((k + \sqrt{n}) \log n)$ am.

- We developed algorithms for:
  - The Semi-Dynamic (SD) case (insertions of points)
  - The Fully-Dynamic (FD) case (insertions and deletions of points)

Contribution

Algorithm	Space	Preprocessing Cost (worst-case)	Query Cost (worst-case)	Update Cost
SD/k-list	$O(n)$	$O(n \log n)$	$O(k)$	$O(\log^2 n + k^2 \log n)$ w.c.
SD/1-list	$O(n)$	$O(n \log n)$	$O(k \log n)$	$O(\log^2 n + k \log n)$ w.c.
FD/k-list	$O(n)$	$O(n \log n)$	$O(k)$	$O((k + \sqrt{n})k \log n)$ am.
FD/1-list	$O(n)$	$O(n \log n)$	$O(k \log n)$	$O((k + \sqrt{n}) \log n)$ am.

- We developed algorithms for:
  - The Semi-Dynamic (SD) case (insertions of points)
  - The Fully-Dynamic (FD) case (insertions and deletions of points)
- For each of the two settings we provide two solutions:
  - k-list focuses on fast query time
  - 1-list focuses on improved update time

Contribution

Algorithm	Space	Preprocessing Cost (worst-case)	Query Cost (worst-case)	Update Cost
SD/k-list	$O(n)$	$O(n \log n)$	$O(k)$	$O(\log^2 n + k^2 \log n)$ w.c.
SD/1-list	$O(n)$	$O(n \log n)$	$O(k \log n)$	$O(\log^2 n + k \log n)$ w.c.
FD/k-list	$O(n)$	$O(n \log n)$	$O(k)$	$O((k + \sqrt{n}) k \log n)$ am.
FD/1-list	$O(n)$	$O(n \log n)$	$O(k \log n)$	$O((k + \sqrt{n}) \log n)$ am.

- We developed algorithms for:
  - The Semi-Dynamic (SD) case (insertions of points)
  - The Fully-Dynamic (FD) case (insertions and deletions of points)
- For each of the two settings we provide two solutions:
  - k-list focuses on fast query time
  - 1-list focuses on improved update time
- Our work focuses on 2-dimensional data
  - Provides a deeper understanding of the complexity of the problem
  - Many applications are inherently 2-dimensional

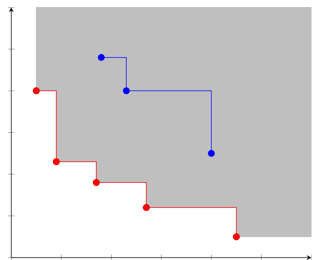
Contribution

Algorithm	Space	Preprocessing Cost (worst-case)	Query Cost (worst-case)	Update Cost
SD/k-list	$O(n)$	$O(n \log n)$	$O(k)$	$O(\log^2 n + k^2 \log n)$ w.c.
SD/1-list	$O(n)$	$O(n \log n)$	$O(k \log n)$	$O(\log^2 n + k \log n)$ w.c.
FD/k-list	$O(n)$	$O(n \log n)$	$O(k)$	$O((k + \sqrt{n})k \log n)$ am.
FD/1-list	$O(n)$	$O(n \log n)$	$O(k \log n)$	$O((k + \sqrt{n}) \log n)$ am.

- We developed algorithms for:
  - The Semi-Dynamic (SD) case (insertions of points)
  - The Fully-Dynamic (FD) case (insertions and deletions of points)
- For each of the two settings we provide two solutions:
  - k-list focuses on fast query time
  - 1-list focuses on improved update time
- Our work focuses on 2-dimensional data
  - Provides a deeper understanding of the complexity of the problem
  - Many applications are inherently 2-dimensional
- The user-defined parameter k is fixed
  - It is defined only once, in the construction phase of the data structures
  - k is generally small compared to the size of the dataset ($k << n$)

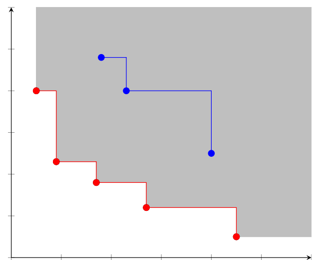
Two-Dimensional Layers of Minima

- The answer set of a skyline query forms the first layer of minima



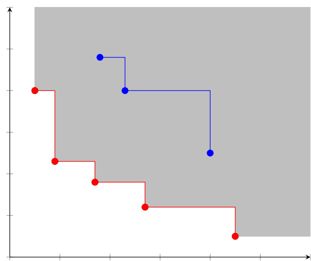
Two-Dimensional Layers of Minima

- The answer set of a skyline query forms the first layer of minima
- We remove the answer set from the dataset and execute a new skyline query on the resulting dataset



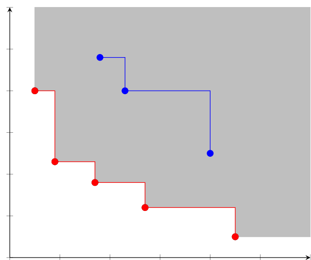
Two-Dimensional Layers of Minima

- The answer set of a skyline query forms the first layer of minima
- We remove the answer set from the dataset and execute a new skyline query on the resulting dataset
- The answer of the new query is the second layer of minima



Two-Dimensional Layers of Minima

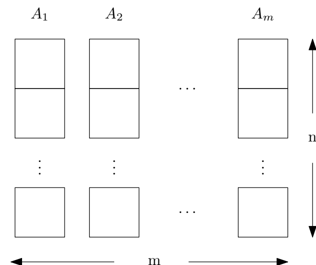
- The answer set of a skyline query forms the first layer of minima
- We remove the answer set from the dataset and execute a new skyline query on the resulting dataset
- The answer of the new query is the second layer of minima
- The set that results from the *i*-th query forms the *i*-th layer of minima
- By collecting all the layers, we form the *layers of minima* of the dataset



Reporting Lemma

- We will use a lemma obtained by Frederickson and Johnson [JCSS'82]

- Let $A_1, \ldots, A_m$ be m sorted arrays
- Given an integer $\mathcal{L} \leq \sum_{i=1}^{m} |A_i|$

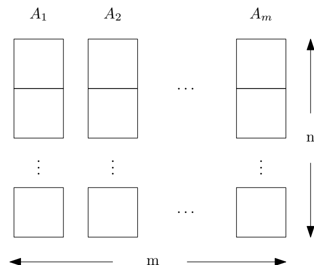


Reporting Lemma

- We will use a lemma obtained by Frederickson and Johnson [JCSS'82]

- Let $A_1, \ldots, A_m$ be m sorted arrays
- Given an integer $\mathcal{L} \leq \sum_{i=1}^{m} |A_i|$

- We can find in $O(m)$ time a value τ that is greater than at least $\mathcal{L}$ but at most $O(\mathcal{L})$ values



Semi-Dynamic Top-k Dominating Queries

- We seek to retrieve the top-k dominating points while supporting insertions of new points

Semi-Dynamic Top-k Dominating Queries

- We seek to retrieve the top-k dominating points while supporting insertions of new points
- An insertion may change the score of many (or even all) points
- We only update (lazily) the score of groups of points that are candidates for being in the final answer

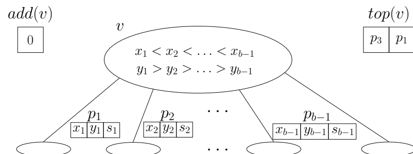
Semi-Dynamic Top-k Dominating Queries

- We seek to retrieve the top-k dominating points while supporting insertions of new points

- An insertion may change the score of many (or even all) points
- We only update (lazily) the score of groups of points that are candidates for being in the final answer

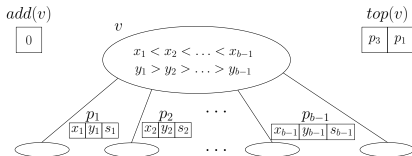
- If a point p dominates a point q, the score of p is strictly greater than the score of q
- The top-k dominating points are located in the first k layers of minima

The Augmented (a,b)-Tree



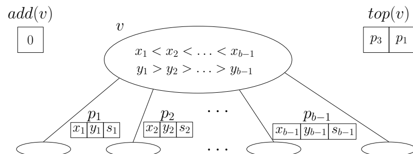
- We model each layer of minima using an augmented leaf-oriented (a, b)-tree built on the layer's points (a,b $= O(1)$)
- The points in a layer are totally ordered on each dimension

The Augmented (a,b)-Tree



- We model each layer of minima using an augmented leaf-oriented (a, b)-tree built on the layer's points (a,b $= O(1)$)
- The points in a layer are totally ordered on each dimension
- We augment a node v with two fields:
  - $add(v)$ is the score that must be added to all points in v's subtree
  - $top(v)$ stores the (sorted) top-k dominating points in v's subtree

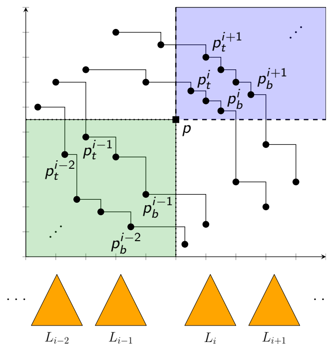
The Augmented (a,b)-Tree



- We model each layer of minima using an augmented leaf-oriented (a, b)-tree built on the layer's points (a,b $= O(1)$)
- The points in a layer are totally ordered on each dimension
- We augment a node v with two fields:
  - $add(v)$ is the score that must be added to all points in v's subtree
  - $top(v)$ stores the (sorted) top-k dominating points in v's subtree
- Insert, delete, split, concatenate in $O(k \log m)$ time and search in $O(\log m)$ time
- The data structure achieves linear space cost

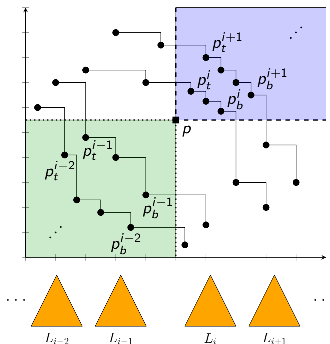
Insertion Overview

1. Compute the score of p
   - 2D Dynamic range counting in $O(\log^2 n)$ time [Chazelle-JComp'88]



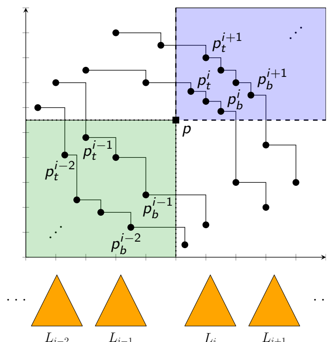
Insertion Overview

1. Compute the score of p
   - 2D Dynamic range counting in $O(\log^2 n)$ time [Chazelle-JComp'88]
2. Find where p must be inserted (L_i)



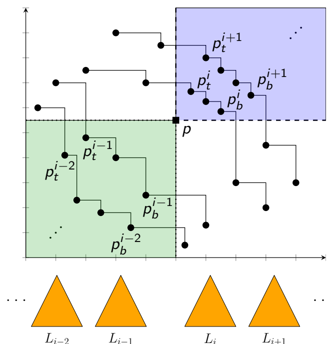
Insertion Overview

1. Compute the score of p
   - 2D Dynamic range counting in $O(\log^2 n)$ time [Chazelle-JComp'88]
2. Find where p must be inserted (L_i)
3. Update scores from L_1 to L_{i-1}



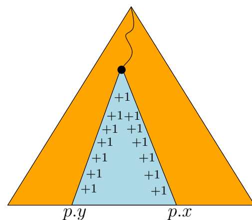
Insertion Overview

1. Compute the score of p
   - 2D Dynamic range counting in $O(\log^2 n)$ time [Chazelle-JComp'88]
2. Find where p must be inserted (L_i)
3. Update scores from L_1 to L_{i-1}
4. Update the layers' structure from L_i to L_k



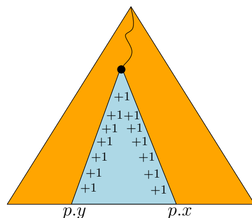
Insertion - Score updating

- Search the layer's tree for *p.x*, *p.y*
- Add +1 to the *add* field of every node hanging "inside" the two search paths
- The top-*k* points (and their relative order) in a subtree do not change



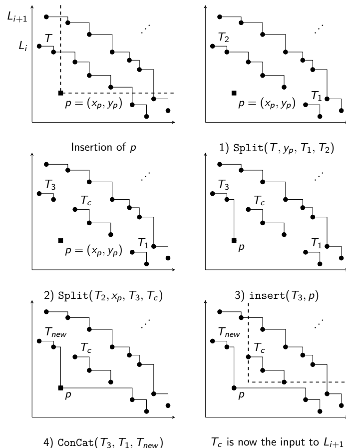
Insertion - Score updating

- Search the layer's tree for *p.x*, *p.y*
- Add +1 to the *add* field of every node hanging "inside" the two search paths
- The top-*k* points (and their relative order) in a subtree do not change
- Moving bottom-up, recompute the *top* field of each node in the search paths
- This is done by taking into account the (updated) *add* fields in the respective nodes



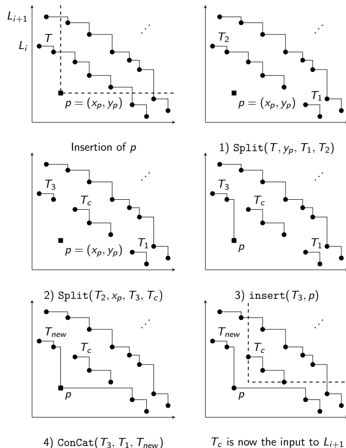
Insertion - Layer restructuring

- Insertion causes cascading changes to the layers' structure
- To update the structure of layers, we perform a series of $O(1)$ tree splits/concatenations (each requiring $O(k \log n)$ time)



Insertion - Layer restructuring

- Insertion causes cascading changes to the layers' structure
- To update the structure of layers, we perform a series of $O(1)$ tree splits/concatenations (each requiring $O(k \log n)$ time)
- On each of the k layers we either perform score updating or layer restructuring
- We obtain the final time bounds by adding the cost of maintaining the range counting data structure



Query

- The *top* field in each tree's root contains the top-k dominating points of that layer
- Apply the Reporting Lemma on all the *top* fields (requires $O(k)$ time)

Query

- The *top* field in each tree's root contains the top-k dominating points of that layer
- Apply the Reporting Lemma on all the *top* fields (requires $O(k)$ time)

- Obtain the score of the k-th top dominating point
- Traverse the (sorted) *top* fields reporting all points with a larger score

Query

- The *top* field in each tree's root contains the top-k dominating points of that layer
- Apply the Reporting Lemma on all the *top* fields (requires $O(k)$ time)

- Obtain the score of the k-th top dominating point
- Traverse the (sorted) *top* fields reporting all points with a larger score

- All of the above require $O(k)$ time in total

Reducing the Update Cost

- Instead of k-sized *top* fields we use 1-sized *top* fields
- Insertion cost is reduced but the query algorithm must change

Reducing the Update Cost

- Instead of k-sized *top* fields we use 1-sized *top* fields
- Insertion cost is reduced but the query algorithm must change
- We build a Strict Fibonacci Heap with all the *top* fields in the k layers
  - Strict Fibonacci Heaps support insertions in $O(1)$ w.c. time and deletions of the maximum key in $O(\log n)$ w.c. time

Reducing the Update Cost

- Instead of k-sized *top* fields we use 1-sized *top* fields
- Insertion cost is reduced but the query algorithm must change

- We build a Strict Fibonacci Heap with all the *top* fields in the k layers
  - Strict Fibonacci Heaps support insertions in $O(1)$ w.c. time and deletions of the maximum key in $O(\log n)$ w.c. time
- When a point p is retrieved from the heap, add all $O(b \log n)$ points found in the *top* fields of the nodes in the search path for p
- To avoid duplicates we employ a marking process

Reducing the Update Cost

- Instead of k-sized *top* fields we use 1-sized *top* fields
- Insertion cost is reduced but the query algorithm must change
- We build a Strict Fibonacci Heap with all the *top* fields in the k layers
  - Strict Fibonacci Heaps support insertions in $O(1)$ w.c. time and deletions of the maximum key in $O(\log n)$ w.c. time
- When a point p is retrieved from the heap, add all $O(b \log n)$ points found in the *top* fields of the nodes in the search path for p
- To avoid duplicates we employ a marking process
- These results can be extended to the word-RAM model of computation

Fully-Dynamic Top-k Dominating Queries

- A deletion is similar to an insertion but instead of adding $+1$ to the *add* fields we add -1

Fully-Dynamic Top-k Dominating Queries

- A deletion is similar to an insertion but instead of adding $+1$ to the *add* fields we add -1
- However, the fully-dynamic setting requires all layers of the dataset (in the worst case)
  - e.g. A deletion from L_k may force a deletion from L_{k+1} and so on

Fully-Dynamic Top-k Dominating Queries

- A deletion is similar to an insertion but instead of adding $+1$ to the *add* fields we add -1
- However, the fully-dynamic setting requires all layers of the dataset (in the worst case)
  - e.g. A deletion from L_k may force a deletion from L_{k+1} and so on
- We perform operations in $k + \sqrt{(}n)$ layers (instead of k)

Fully-Dynamic Top-k Dominating Queries

- A deletion is similar to an insertion but instead of adding $+1$ to the *add* fields we add -1
- However, the fully-dynamic setting requires all layers of the dataset (in the worst case)
  - e.g. A deletion from L_k may force a deletion from L_{k+1} and so on
- We perform operations in $k + \sqrt{(}n)$ layers (instead of k)
- After $\sqrt{(}n)$ deletions only k layers remain "valid" so we rebuild the entire data structure

Fully-Dynamic Top-k Dominating Queries

- A deletion is similar to an insertion but instead of adding $+1$ to the *add* fields we add -1
- However, the fully-dynamic setting requires all layers of the dataset (in the worst case)
  - e.g. A deletion from L_k may force a deletion from L_{k+1} and so on
- We perform operations in $k + \sqrt{(}n)$ layers (instead of k)
- After $\sqrt{(}n)$ deletions only k layers remain "valid" so we rebuild the entire data structure
- The reduced update cost methods can also be applied in the fully-dynamic setting

Conclusions and Future Work

- Development, for the first time, of algorithms that answer semi-dynamic and fully-dynamic top-k dominating queries in the 2-dimensional space with non-trivial performance guarantees

Conclusions and Future Work

- Development, for the first time, of algorithms that answer semi-dynamic and fully-dynamic top-k dominating queries in the 2-dimensional space with non-trivial performance guarantees
- The parameter k is assumed to be user-defined and fixed between queries

Conclusions and Future Work

- Development, for the first time, of algorithms that answer semi-dynamic and fully-dynamic top-k dominating queries in the 2-dimensional space with non-trivial performance guarantees
- The parameter k is assumed to be user-defined and fixed between queries
- Interesting research directions for future work include:
  - Lowering the update cost in the fully-dynamic case
  - Multi-dimensional top-k dominating queries
  - Top-k dominating queries in the external model
  - Top-k dominating queries under the streaming model

Thank You

Questions?