Efficient Processing of Top-k Dominating Queries on Multi-dimensional Data

M. L. Yiu
Aalborg University

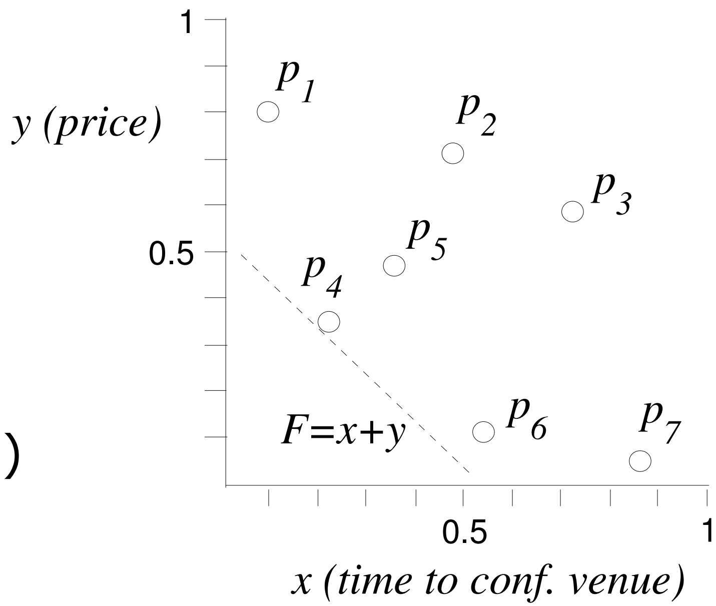
N. Mamoulis
University of Hong Kong

Outline

- Motivations and applications
- Background
- Eager approach
- Lazy approach
- Experimental results
- Conclusions

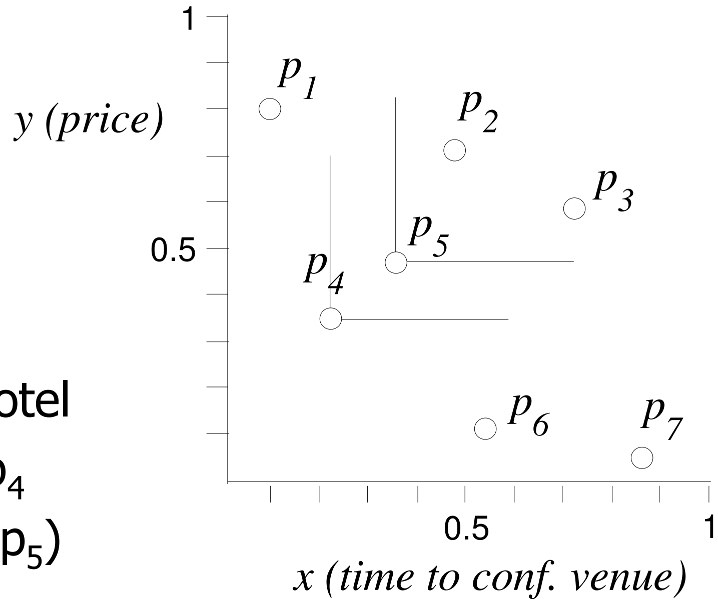
Top-k Query, Skyline Query

- D: set of points in multi-dimensional space $\Re^d$
- Top-k query
  - k points with the lowest F values
  - Top-2: p_4, p_6
  - Require a ranking function ☹
  - Result affected by scales of dimensions ☹
- Skyline query
  - $p > p'$: $(\exists\ i,\ p[i] < p'[i]) \wedge (\forall\ i,\ p[i] \leq p'[i])$
  - Points not dominated by any other point
  - Skyline: p_1, p_4, p_6, p_7
  - Uncontrolled result size ☹



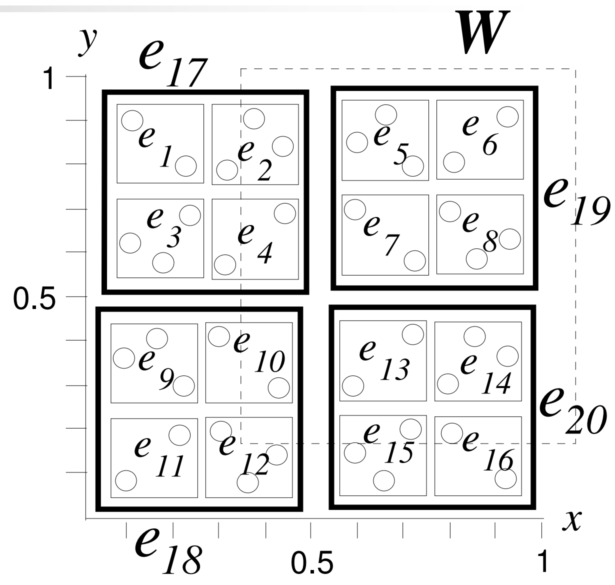
Top-k Dominating Query

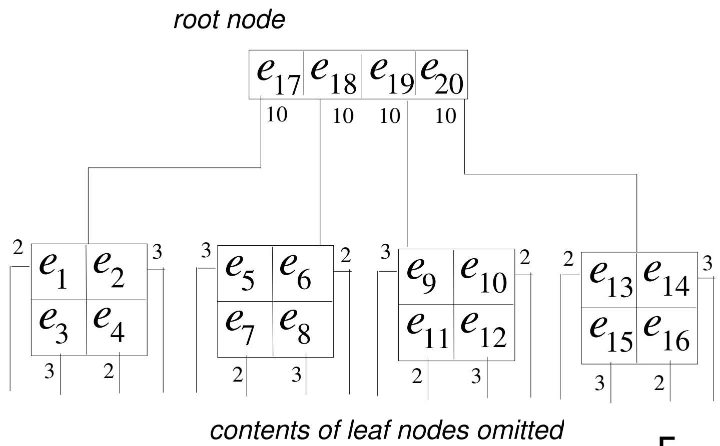
- Intuitive score function: $\mu(p) = | \{ p' \in D, p > p' \} |$
- Top-k dominating query
  - Also called k-dominating query [Papadias et. al. 2005]
  - Returns k points with the highest μ values
  - Top-2 dominating points: p_4 (3), p_5 (2)
- Advantages ☺
  - Control of result size
  - No need to specify ranking function
  - Result independent of scales of dimensions
- Application: decision support
  - The query captures the most `significant' hotel
  - A conference participant attempts to book p_4
  - If p_4 is fully booked, then try the next one (p_5)



Related Work

- Spatial aggregation processing
  - E.g., count the number of points in a region
  - Aggregate R-trees [Papadias et. al. 2001]
  - Example: COUNT R-tree
    - Each entry is augmented with the COUNT of points in its subtree
  - Query: find the number of points in W
    - W contains the entry e_{19}
    - Increment the answer by COUNT(e_{19}), without accessing its subtree
    - Augmented values speed up the counting the process



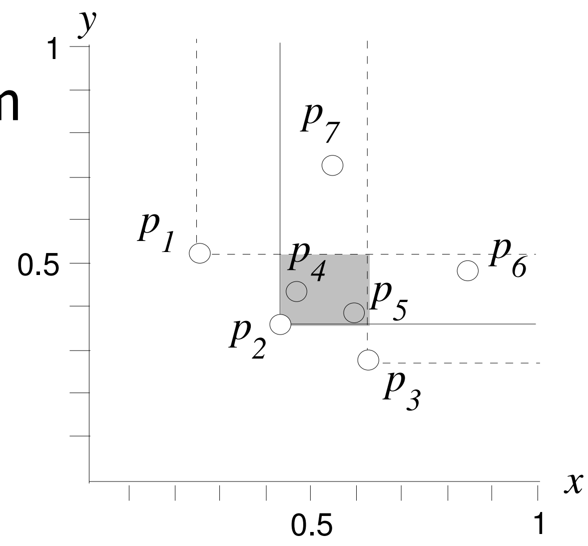


Top-k Dominating Query

- Processing of the top-k dominating query
- Naïve solution: Block Nested Loop join, compute the score of every point
  - Quadratic cost of input size
- Goal: develop efficient algorithm on indexed multi-dimensional points (R-tree)
  - Eager approach
  - Lazy approach

Existing Skyline-based Solution

- [Papadias et. al. 2005] Apply a skyline algorithm iteratively to obtain k-dominating points
- Example: top-2 dominating query
- Iteration 1
  - Property: $\forall$ p,p′$\in$D, p>p′ $\Rightarrow$ μ(p)>μ(p′)
  - Find the skyline points
  - Count their scores (by accessing the tree)
  - Report the first result: p_2 (4)
- Iteration 2
  - Find the constrained skyline (gray region)
    - Region dominated by p_2 but not others (p_1, p_3)
  - Count their scores and compare them with points in all previous iterations
  - Report the next result: p_4 (2)

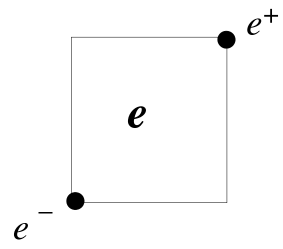


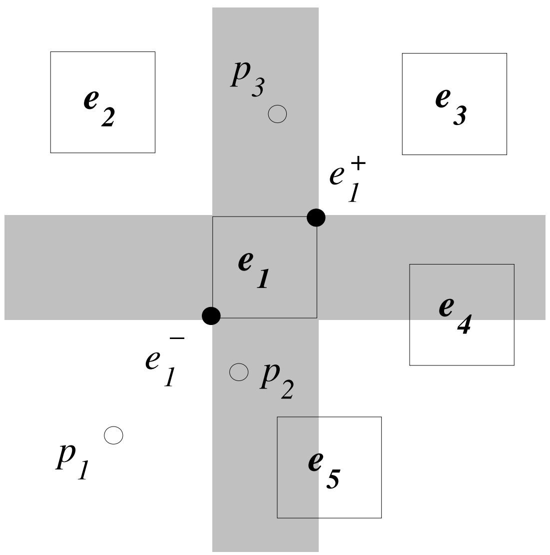
Slow! At large skyline size!

Counting cost □ skyline cost

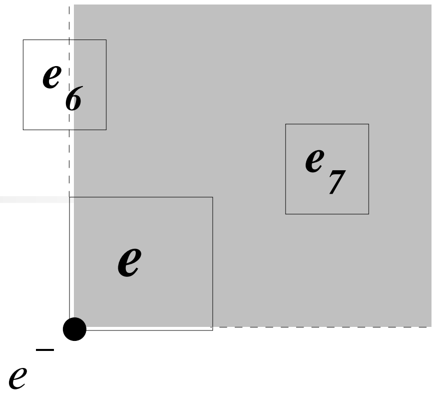
Our Observation

- The counting operation is the most important
  - Index the dataset by a COUNT R-tree
- Corner locations of an entry e
  - Lower corner e^-, upper corner e^+
- Three possible dominance relationships
  - Full dominance: $p_1 \succ e_1^-$
    - p_1 dominates all points in e_1
  - Partial dominance: $p_2 \succ e_1^+$ and $p_2 \nsucc e_1^-$
    - p_2 may dominate some points in e_1
  - No dominance: $p_3 \nsucc e_1^+$
    - p_3 dominates no points in e_1
- Similar dominance relationships between entries
  - e_1 fully dominates e_3
  - e_1 partially dominates e_4





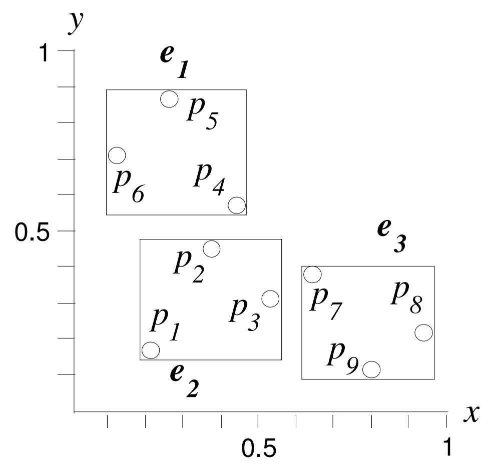
Our Eager Approach

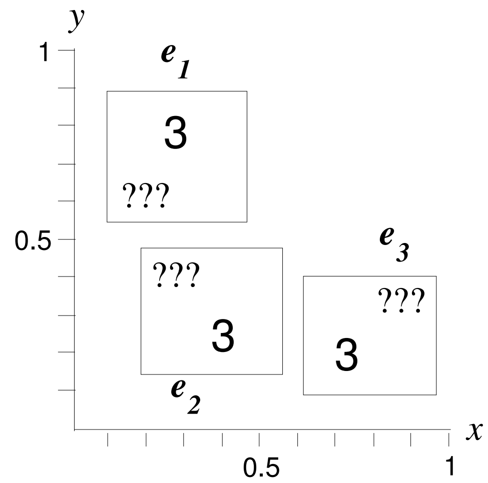


- **Tight-most** upper-bound score of an entry e: $\mu(e^-)$
  - Tight-most in the sense that the subtree content of e is not used
  - Compute $\mu(e^-)$ by visiting nodes in the tree
- Traverse the nodes in the tree, in **descending order** of their upper bound scores
  - Use a max-heap H for organizing the entries to be visited in descending order of their *upper bound* scores
  - For each encountered entry e, compute its $\mu(e^-)$ immediately ← eager
  - Keep the best-k points (with the highest scores) found so far
  - Terminates when the top entry of H has upper-bound score smaller than the current best-k points
- No need to compute the whole skyline!

Tight-most Upper-bound Score Necessary?

- It suffices to derive a loose upper-score bound $\mu^u(e)$, for a non-leaf entry e
- Eager algorithm is correct, as long as $\mu^u(e) \geq \mu(e^-)$
- Develop the **lightweight counting** technique to compute $\mu^u(e)$, without accessing leaf nodes
  - Based on dominance relationships between entries
  - Much lower cost, relatively tight bound ☺
- Comparison on the example
  - Tight-most bounds: $\mu(e_1^-)=3$, $\mu(e_2^-)=7$, $\mu(e_3^-)=3$
  - Loose bounds: $\mu^u(e_1)=3$, $\mu^u(e_2)=9$, $\mu^u(e_3)=3$
  - The child node of e_2 will still be accessed first
  - Ordering of entries approximately preserved (i.e., effective search ordering) ☺





Our Lazy Approach

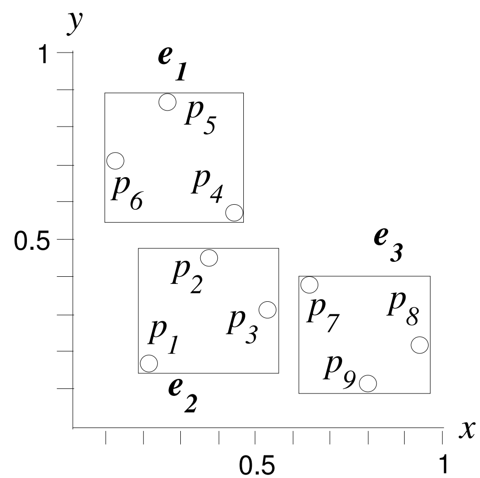
- Problem of the Eager approach
  - Some tree nodes may be visited multiple times (due to explicit counting of upper score bounds of entries)
- We then propose a Lazy approach
  - Visit each tree node at most **ONCE**!
  - Maintain lower $\mu^l(e)$ bound and upper $\mu^u(e)$ bound for each visited entry, initially $\mu^l(e)=0$ and $\mu^u(e)=N$
  - When a node is accessed, we **refine** the bounds of visited entries

Lazy Approach: Example

- Traversal order: assume that the node with highest upper bound is visited first
- Update bounds only based on visited entries
- Access root node
  - $\mu(e_1)=[0,3]$, $\mu(e_2)=[0,9]$, $\mu(e_3)=[0,3]$
  - $S=\{e_1, e_2, e_3\}$
- Access the child node of e_2
  - $\mu(p_1)=[1,7]$, $\mu(p_2)=[0,3]$, $\mu(p_3)=[0,3]$
  - Score bounds of e_3 unchanged
- $S=\{e_3, p_1, p_2, p_3\}$
- ……

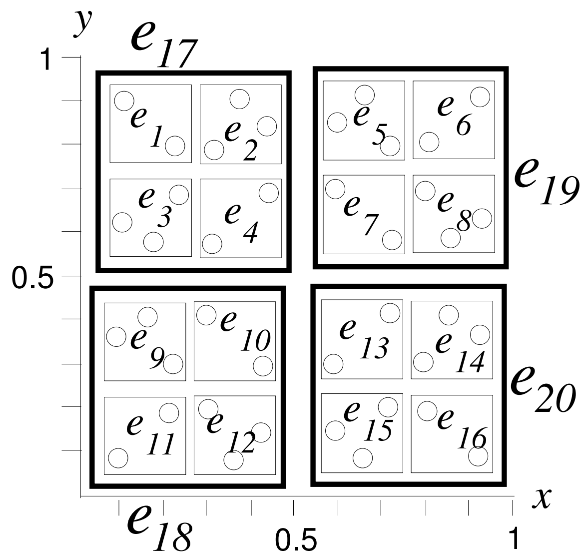
[1] e fully dominates e'
→ $\mu^l(e)$ and $\mu^u(e)$ both added by COUNT(e')

[2] e partially dom. e':
→ only $\mu^u(e)$ added by COUNT(e')

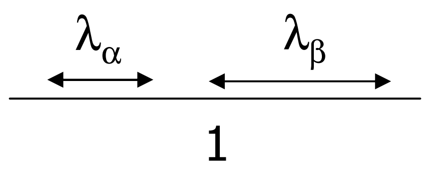


Traversal Order of Lazy Approach

- Performance of Lazy depends on its traversal order
- Intuitive order: choose the **non-leaf** entry (in S) with the highest upper bound score $\mu^u(e)$
- Is this really the best traversal order?
- Example
  - Access ordering: root, e_{18}, ...
  - S={e_{17}, e_{19}, e_{20}, e_{11}, e_{12}, e_9, e_{10}}
  - Current score bounds of e_{11}
    - Upper bound=40
    - Lower bound=10+2=12 (low, due to partial dominance)
    - Current best score=12, only few entries can be pruned!
- Objective of search
  - Examine entries of large upper bounds early
  - Eliminate partial dominance relationships of entries in S



Analysis of Partial Dominance



- Assume that α and β are two entries
- Let λ_α be the length projection of α along a dimension
- Pr(α and β do not intersect along a given dimension τ)
  $= 1 - (\lambda_\alpha + \lambda_\beta)$
- Pr(α and β have partial dominance relationship)
  = Pr(α and β intersect at least one dimension)
  $= 1 - (1 - (\lambda_\alpha + \lambda_\beta))^d$, where d is the number of dimensions
- Observation: the above probability is low when $(\lambda_\alpha + \lambda_\beta)$ is small, i.e., both α and β are at low levels
- A better traversal ordering
  - Find non-leaf entries (in S) with the highest level
  - Among them, choose the one with the highest upper bound score

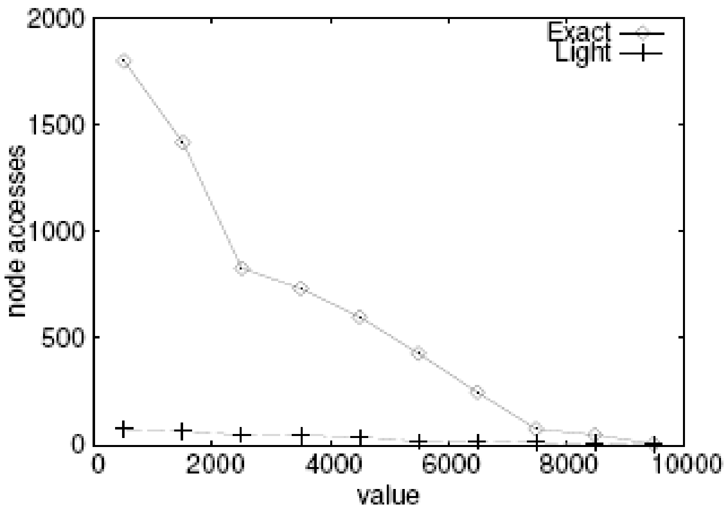
Experiments on Synthetic Data

- Algorithms
  - ITD (Existing **Skyline-based** method, plus optimizations)
  - LCG (**Eager** approach, with lightweight counting)
  - CBT (**Lazy** approach, with our novel traversal order)
- Synthetic datasets
  - UI (independent), CO (correlated), AC (anti-correlated)
- Default parameters values
  - Node page size of COUNT R-tree : 4K bytes
  - LRU buffer size (%): **5**
  - Datasize N (million): **1**
  - Data dimensionality d: **3**
  - Result size k: **16**

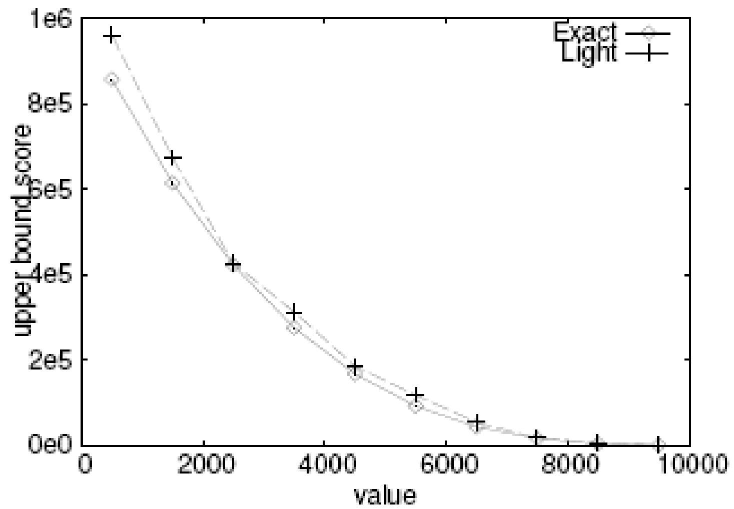
Counting Technique in Eager

Compare the computation of
exact upper-bound score and
loose upper-bound score

Uniform data



Node accesses



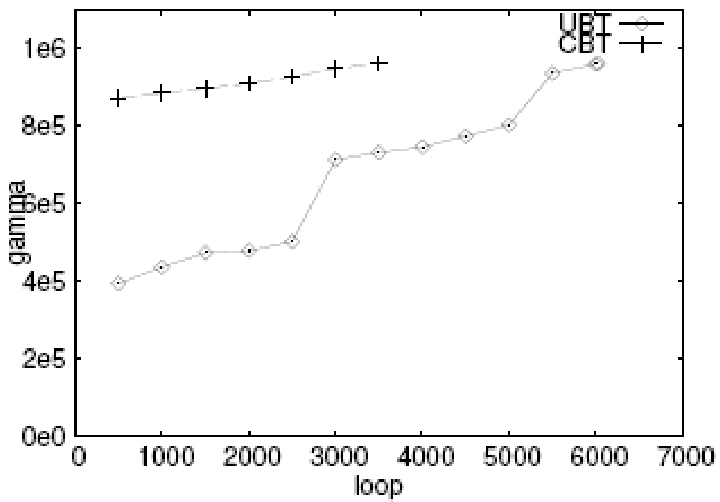
Upper-bound score of the entry

value ~ location of the entry e

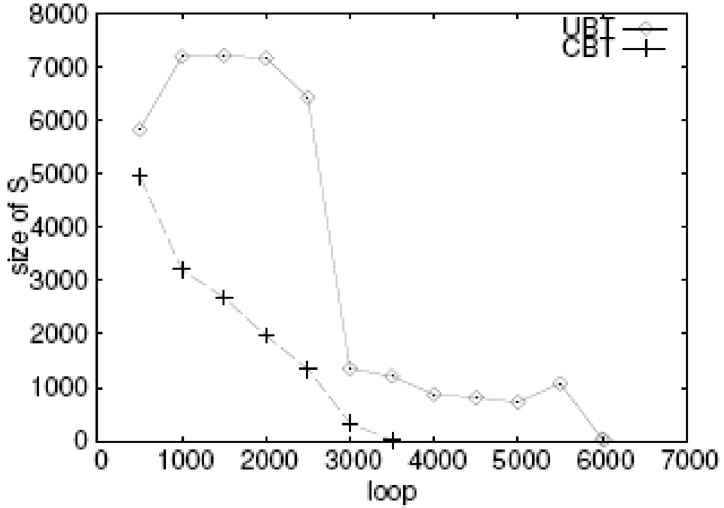
Traversal Order in Lazy

Compare the traversal of
upper-bound order and
novel order

Uniform data

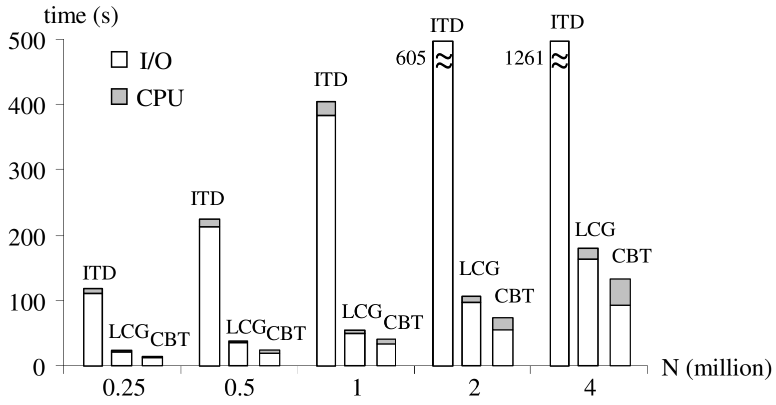


Value of γ
(best score of a point)

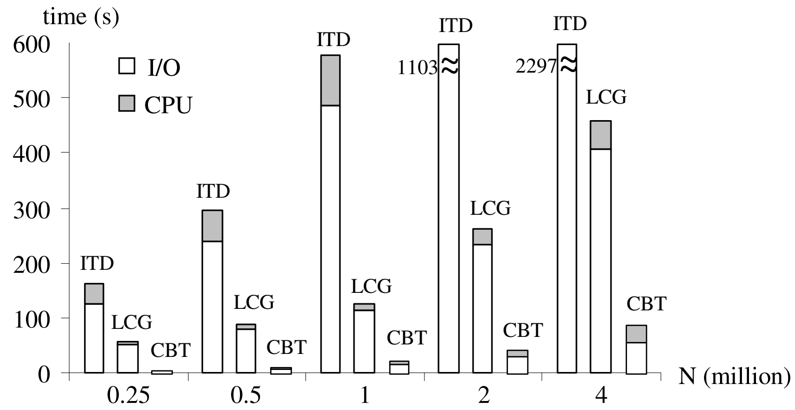


Size of S
(number of existing entries in memory)

I/O cost vs N

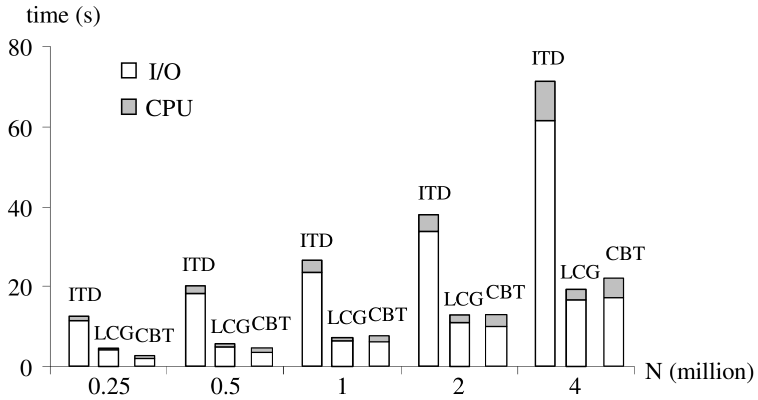


UI data



AC data

CO data



Application of Top-k Dominating Points

- Real datasets (sports statistics)
  - NBA: 19112 players; BASEBALL: 36898 pitchers
- Apply top-k dominating queries to discover "top" players, without using any expert knowledge
- Results match the public's view of super-star players in NBA and BASEBALL

Identified by player name & year

Attributes

Top-5 dominating points

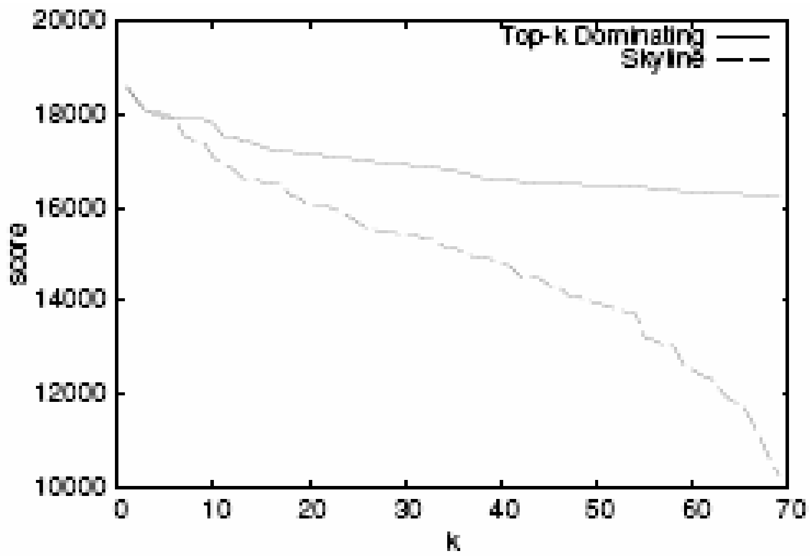
Score	NBA Player / Year	gp	pts	reb	ast
18585	Wilt Chamberlain / 1967	82	1992	1952	702
18299	Billy Cunningham / 1972	84	2028	1012	530
18062	Kevin Garnett / 2002	82	1883	1102	495
18060	Julius Erving / 1974	84	2343	914	462
17991	Kareem Abdul-Jabbar / 1975	82	2275	1383	413

Score	BASEBALL Pitcher / Year	w	g	sv	so
34659	Ed Walsh / 1912	27	62	10	254
34378	Ed Walsh / 1908	40	66	6	269
34132	Dick Radatz / 1964	16	79	29	181
33603	Christy Mathewson / 1908	37	56	5	259
33426	Lefty Grove / 1930	28	50	9	209

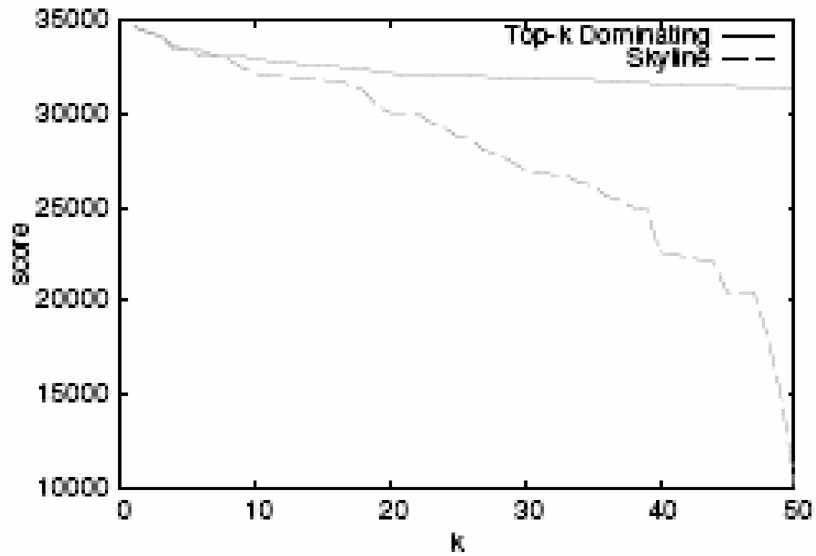
Not skyline points!

Skyline vs Top-k Dominating points

- Perform a skyline query, compute top-k dominating points by setting k to the skyline size (69 for NBA and 50 for BASEBALL)
- Plot their dominating scores in descending order
- Observations
  - Top-k dominating points have much higher scores than skyline points
  - Top-k dominating points are more informative to users



NBA



BASEBALL

Conclusions

- Recognize the importance of top-k dominating query as a data analysis tool
- Our algorithms on R-tree
  - LCG (**Eager** approach, with lightweight counting)
  - CBT (**Lazy** approach, with a novel traversal order)
- CBT has the best performance, relatively stable performance across different data distribution
- Future work
  - For non-indexed data, algorithms based on hashing
  - Approximate top-k dominating result, with error guarantee

References

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

[Papadias et. al. 2005] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive Skyline Computation in Database Systems. TODS, 30(1):41–82, 2005.

Alternative solutions?

- Pre-computation possible? ☒
  - Materialize the `score' of every point
  - Updates: change the 'score' of influenced points
  - Update cost is expensive for dynamic datasets
- Approximation by using dominating area? ☒
  - DomArea(p_i) = Area dominated by the point p_i
  - Dominating area cannot provide bounds for μ
    - DomArea(p_1) > DomArea(p_4)
    - but $\mu(p_1)=1 < \mu(p_4)=2$!!!
- Unlike the dominating area, computing μ value (or even its upper bound) requires accessing data
- Related work on skyline
  - Skyline on R-tree: BBS [Papadias et. al. 2005]
    - Best-first traversal (from the origin) of R-tree
    - Keep found skyline points for pruning others