An Optimal and Progressive Algorithm for Skyline Queries

Instructor: Cyrus Shahabi
Outline

• Introduction
  – Skyline queries
  – Existing solutions
  – Motivation
• Algorithms BBS
• Other discussions
• Experiments
• Conclusion
Finding the Cheapest & Closest Hotels

Which one is better?

- $i$ and $h$?
- $i$, because its price and distance \textit{dominate} those of $h$.
- $i$ and $k$?
- We do not know.
Skyline Objects

- A set of objects not dominated by any other object.

- Dominance region
Existing Solutions

- Block Nested Loop (BNL)
- Divide-and-Conquer (D&C)
- Bitmap method
- Index method
- Nearest Neighbor (NN)

Elementary skyline algorithms

Progressive skyline algorithms
Existing Solutions

• **Block Nested Loop (BNL)**
  – Scan the dataset and keep a list of candidate skyline points.
  – Compare a point $p$ with every other point in the list.

• **Advantages**
  – Applicable for any dimensionality
  – Does not need sorting or indexing of data file

• **Disadvantages**
  – Numerous comparisons
  – Inadequacy for on-line processing
Existing Solutions

- **Divide-and-Conquer (D&C)**
  - Divide the dataset into several partitions.
  - Compute partial skylines in each partition.
  - Compute global skylines by merging them.

![Diagram showing skylines](image)
Existing Solutions

- Nearest Neighbor (NN)
  - Find nearest neighbor point -> skyline
  - Prune all the points in the dominance region of this point
  - Divide the space by the nearest neighbor point -> to-do lists
  - Compute recursively until empty space.
Existing Solutions

- NN over three or more dimensions
  - Has overlapped partitions in divided subspaces.
  - Needs duplicate elimination.

NN partitions for 3 dimensions
Motivation

• Advantages of $NN$ algorithm
  – Fast running time to finding the first result
  – Progressiveness

• Disadvantages of $NN$ algorithm
  – Redundant I/O computation
    • Gets worse as dimensionality increases
  – Explosive to-do list size
Contents

• Introduction
• Algorithms BBS
  — Preliminary: R-Tree
  — How BBS works
  — Example
• Other Discussions
• Experiments
• Conclusion
Branched and Bound Skyline (BBS)

- Assume all points are indexed in an R-tree.
- **Top-down Approach**
- \( \text{mindist} = \) the \( L_1 \) distance between its lower-left corner and the origin.

\[
f(x, y) = x + y
\]
Branched and Bound Skyline (BBS)

• Data structure
  – *Heap* sorted by min distance
  – *List* to maintain the current skyline

• Dominance check condition
  – Before expanding, compare to current skylines.
  – Before inserting an object, also check for internal objects.

• Stop condition: empty heap
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
Example of BBS

- Each heap entry keeps the mindist of the MBR.
Example of BBS

- Process entries in ascending order of their mindists.

**action**
- access root
- expand `e7`

**heap contents**
- `<e_7,4><e_6,6>`
- `<e_3,5><e_6,6><e_5,8><e_4,10>`

**S**
- `∅`

**R**
- `e_6 e_7`

**N_6**
- `e_1 e_2`

**N_1**
- `a b c`

**N_2**
- `d e f`

**N_3**
- `g h i`

**N_4**
- `l k`

**N_5**
- `m n`

**N_7**
- `e_3 e_4 e_5`
Example of BBS

action
access root
expand e7
expand e3

heap contents
\(<e_7,4><e_6,6>\)
\(<e_3,5><e_6,6><e_5,8><e_4,10>\)
\(<i,5><e_6,6><e_5,8><e_4,10>\)

\(S\)
\(\emptyset\)
\(\emptyset\)
\({i}\)
Example of BBS

action
access root
expand e7
expand e3
No-insert e2

heap contents
$S$
$\emptyset$
$\emptyset$
$\{i\}$
$\{i\}$
Example of BBS

- **action**
  - access root
  - expand e7
  - expand e3
  - expand e6
  - remove e5

- **heap contents**
  - $S$
    - access root
    - expand e3
    - expand e6
    - remove e5
  - $\emptyset$
  - $\emptyset$
  - $\{i\}$
  - $\{i\}$
  - $\{i\}$

- **R**
  - e6
  - e7

  - N6
    - e1
    - e2

  - N1
    - a
    - b
    - c

  - N2
    - d
    - e
    - f

  - N3
    - g
    - h
    - i

  - N4
    - l
    - k

  - N5
    - m
    - n
Example of BBS

- **Action**
  - Access root
  - Expand e7
  - Expand e3
  - Expand e6
  - Remove e5
  - Expand e1

- **Heap Contents**
  - $S = \emptyset$
  - $\emptyset$
  - $\{i\}$
  - $\{i\}$
  - $\{i\}$
  - $\{i, a\}$
  - $<e_7,4><e_6,6>$
  - $<e_3,5><e_6,6><e_5,8><e_4,10>$
  - $<i,5><e_6,6><e_5,8><e_4,10>$
  - $<e_5,8><e_1,9><e_4,10>$
  - $<e_1,9><e_4,10>$
  - $<a,10><e_4,10>$

- **Diagram**
  - Nodes: $N_1, N_2, N_3, N_4, N_5, N_6, N_7$
  - Elements: $a, b, c, d, e, f, g, h, i, l, k, m, n$
Example of BBS

- **Action**: access root, expand e7, remove e6, expand e5, expand e1, expand e4
- **Heap contents**:
  - $S$
  - $\emptyset$
  - $\{i\}$
  - $\{i\}$
  - $\{i\}$
  - $\{i,a\}$
  - $\{i,a,k\}$
Contents

• Introduction
• Algorithms BBS
• Other Discussions
  – Constrained skyline queries
  – K-dominating queries
• Experiments
• Conclusion
Constrained Skyline Queries

 constrain: $4 \leq y \leq 7$

action  
access root
expand e7

heap contents  
$\langle e7, 4 \rangle, \langle e6, 6 \rangle$
$\langle e3, 5 \rangle, \langle e6, 6 \rangle, \langle e4, 10 \rangle$

$S$
$\emptyset$

$R$
$e_6 \quad e_7$

$N_6$
$e_1 \quad e_2$
$a \quad b \quad c$

$N_1$

d \quad e \quad f$

$N_2$

g \quad h \quad i$

$N_3$

l \quad k$

$N_4$

m \quad n$

$N_5$

$N_7$
Constrained Skyline Queries

**constrain**: $4 \leq y \leq 7$

- **action**
  - access root
  - expand $e_7$
  - expand $e_3$

- **heap contents**
  - $<e_7, 4>, <e_6, 6>$
  - $<e_3, 5>, <e_6, 6>, <e_4, 10>$
  - $<e_6, 6>, <e_4, 10>, <g, 11>$

- **$S$**
  - $\emptyset$
  - $\emptyset$
  - $\emptyset$

**Diagram**

The diagram illustrates the constrained skyline queries with nodes labeled $a$ to $m$ and corresponding coordinates and actions are marked on the nodes and edges of the graph. The nodes are connected in a hierarchical structure, and the actions performed are indicated by the labels on the edges.
Constrained Skyline Queries

constraint: $4 \leq y \leq 7$
Constrained Skyline Queries

constrain: \(4 \leq y \leq 7\)
Constrained Skyline Queries

\[ 4 \leq y \leq 7 \]

**action** | **heap contents** | **S**
---|---|---
access root | \(<e7, 4>, <e6, 6>\) | \(\emptyset\)
expand e7 | \(<e3, 5>, <e6, 6>, <e4, 10>\) | \(\emptyset\)
expand e3 | \(<e6, 6>, <e4, 10>, <g, 11>\) | \(\emptyset\)
expand e6 | \(<e4, 10>, <g, 11>, <e2, 11>\) | \(\emptyset\)
expand e4 | \(<g, 11>, <e2, 11>, <l, 14>\) | \{g\}
expand e2 | \(<f, 12>, <d, 13>, <l, 14>\) | \{g, f, l\}
K-dominating Queries

- Retrieve 3 points that dominate the largest number of other points.

- $\text{num}(i) = 9$, $\text{num}(a) = 2$, $\text{num}(k) = 2$

- $\text{h and m may dominate at most 7 points. (num}(i) - 2)$
K-dominating Queries

num(h) = 7, num(m) = 5, num(a) = 2, num(k) = 2

2-dominating result: \{i, h\}

3-dominating result: \{i, h, m\}

c and g may dominate at most 5 points. (num(h) – 2)
Contents

• Introduction
• Algorithms BBS
• Other Discussions
• Experiments
• Conclusion
Experiments (Comparing BBS with NN)

• Datasets:
  – Independent (uniform), anti-correlated

• Dimensionality:
  – In range [2,5]

• Cardinality:
  – In range [100k,10M]

• Machine:
  – Pentium 4 CPU
  – 2.4 GHz
  – 512MB Ram
EXP 1: Effect of dimensionality

Figure 5.1: Node accesses vs. $d$ ($N=1M$)

Figure 5.2: CPU-time vs. $d$ ($N=1M$)
EXP 1: Observations

- NN could not terminate successfully for $d > 4$ (independent), $d > 3$ (anti-correlated)
  - Due to prohibitive size of the *to-do* lists

- Degradation of NN is caused mainly by
  - Growth of the number of partitions
  - Growth of number of duplicates

- Degradation of BBS is due to
  - Growth of skyline points
  - Poor performance of R-tree in higher dimensions
EXP 2: Effect of cardinality

**Figure 5.5:** Node accesses vs. $N (d=3)$

**Figure 5.6:** CPU-time vs. $N (d=3)$
EXP 3: Progressive behavior

Figure 5.7: Node accesses vs. # points returned ($N=1M$, $d=3$)

Figure 5.8: CPU-time vs. # points returned ($N=1M$, $d=3$)
Experiments - Key Notes

• In General BBS outperforms NN significantly
• For dimensionality
  – NN does not terminate when d increases due to explosive to do list
• For cardinality
  – NN does not terminate here as well if N>=5M due to to do list
• For Progressive behavior
  – NN requires more node access and CPU time to return number of reported points
Contents

- Introduction
- Algorithms BBS
- Other Discussions
- Experiments
- Conclusion
Conclusion and Future Work

• All existing database algorithms for skyline computation have several deficiencies.
• BBS overcomes all these deficiencies
  – it is efficient for both progressive and complete skyline computation, independently of the data characteristics
  – it can easily handle numerous alternative skyline queries (e.g. constrained, k-dominating)
  – it can be used for any subset of the dimensions
  – it has limited main memory requirements
• Future work
  – Investigate alternatives for high dimensional spaces where R-Trees are inefficient.
  – Approximate skyline queries
References

1. Papadias, D.; Tao, Y.; Fu, G. & Seeger, B. 

2. A presentation by Ali Khodaei in csci587 Fall’2010