Maximal Vector Computation

in Large Data Sets

Parke Godfrey\textsuperscript{1} \quad Ryan Shipley\textsuperscript{2} \quad Jarek Gryz\textsuperscript{1}

\textsuperscript{1}York University
Toronto, CANADA

\textsuperscript{2}College of William and Mary
Williamsburg, USA

30 August 2005

VLDB

Trondheim, Norway
I. Introduction

What is Skyline?

- an extension to SQL
- filtering for the Pareto-optimal tuples
- a way to express “best-match” & preference queries

```
select . . .
from . . .
where . . .
group by . . .
skyline of D₁ [min | max | diff], . . .,
        Dₖ [min | Max | diff]
having . . .
```

[Börzsönyi, Kossmann, & Stocker 2001 (ICDE)]
I. Introduction

What is Skyline?

- an extension to SQL
- filtering for the *Pareto-optimal* tuples
- a way to express “best-match” & preference queries

```
select . . .
from . . .
where . . .
group by . . .
   skyline of D₁ [min | max | diff], . . .,
   Dₖ [min | Max | diff]
having . . .
```

[Börzsönyi, Kossmann, & Stocker 2001 (ICDE)]

- Have been ~30 skyline-related papers in DB-related journals, conferences, & workshops since.
- Next two talks are on skyline, & one at PhD Workshop.
Consider a **Hotel** table with columns name, address, dist (distance to the beach), stars (quality rating), & price.

select name, address
from Hotel
skyline of stars max,
dist min,
price min
Consider a **Hotel** table with columns name, address, dist (distance to the beach), stars (quality rating), & price.

```sql
SELECT name, address
FROM Hotelskyline
WHERE
  stars = (SELECT MAX(stars) FROM Hotelskyline)
  AND dist = (SELECT MIN(dist) FROM Hotelskyline)
  AND price = (SELECT MIN(price) FROM Hotelskyline)
```

<table>
<thead>
<tr>
<th>name</th>
<th>stars</th>
<th>dist</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aga</td>
<td>⭐⭐</td>
<td>0.7</td>
<td>1,175</td>
</tr>
<tr>
<td>Fol</td>
<td>⭐</td>
<td>1.2</td>
<td>1,237</td>
</tr>
<tr>
<td>Kaz</td>
<td>⭐</td>
<td>0.2</td>
<td>750</td>
</tr>
<tr>
<td>Neo</td>
<td>⭐⭐⭐</td>
<td>0.2</td>
<td>2,250</td>
</tr>
<tr>
<td>Tor</td>
<td>⭐⭐⭐</td>
<td>0.5</td>
<td>2,550</td>
</tr>
<tr>
<td>Uma</td>
<td>⭐⭐</td>
<td>0.5</td>
<td>980</td>
</tr>
</tbody>
</table>
Consider a **Hotel** table with columns **name**, **address**, **dist** (distance to the beach), **stars** (quality rating), & **price**.

```sql
select name, address
from Hotel
skyline of stars max,
dist min,
price min
```

<table>
<thead>
<tr>
<th>name</th>
<th>stars</th>
<th>dist</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aga</td>
<td>★★</td>
<td>0.7</td>
<td>1,175</td>
</tr>
<tr>
<td>Fol</td>
<td>★</td>
<td>1.2</td>
<td>1,237</td>
</tr>
<tr>
<td>Kaz</td>
<td>★</td>
<td>0.2</td>
<td>750</td>
</tr>
<tr>
<td>Neo</td>
<td>★★★</td>
<td>0.2</td>
<td>2,250</td>
</tr>
<tr>
<td>Tor</td>
<td>★★★</td>
<td>0.5</td>
<td>2,550</td>
</tr>
<tr>
<td>Uma</td>
<td>★★</td>
<td>0.5</td>
<td>980</td>
</tr>
</tbody>
</table>
```

- **Blue**: currently considering
- **Green**: “trumps” current
- **Red**: skyline
- **Gray**: not skyline
A Skyline Example

Consider a **Hotel** table with columns **name**, **address**, **dist** (distance to the beach), **stars** (quality rating), & **price**.

```sql
select name, address
from Hotel
skyline of stars max, dist min, price min
```

<table>
<thead>
<tr>
<th>name</th>
<th>stars</th>
<th>dist</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aga</td>
<td><strong>☆</strong></td>
<td>0.7</td>
<td>1,175</td>
</tr>
<tr>
<td>Fol</td>
<td>*</td>
<td>1.2</td>
<td>1,237</td>
</tr>
<tr>
<td>Kaz</td>
<td>*</td>
<td>0.2</td>
<td>750</td>
</tr>
<tr>
<td>Neo</td>
<td>***</td>
<td>0.2</td>
<td>2,250</td>
</tr>
<tr>
<td>Tor</td>
<td>***</td>
<td>0.5</td>
<td>2,550</td>
</tr>
<tr>
<td>Uma</td>
<td><strong>☆</strong></td>
<td>0.5</td>
<td>980</td>
</tr>
</tbody>
</table>
```

- currently considering
- “trumps” current
- skyline
- not skyline
A Skyline Example

Consider a **Hotel** table with columns **name**, **address**, **dist** (distance to the beach), **stars** (quality rating), & **price**.

<table>
<thead>
<tr>
<th>name</th>
<th>stars</th>
<th>dist</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aga</td>
<td>★★★</td>
<td>0.7</td>
<td>1,175</td>
</tr>
<tr>
<td>Fol</td>
<td>★</td>
<td>1.2</td>
<td>1,237</td>
</tr>
<tr>
<td>Kaz</td>
<td>★</td>
<td>0.2</td>
<td>750</td>
</tr>
<tr>
<td>Neo</td>
<td>★★★★</td>
<td>0.2</td>
<td>2,250</td>
</tr>
<tr>
<td>Tor</td>
<td>★★★★</td>
<td>0.5</td>
<td>2,550</td>
</tr>
<tr>
<td>Uma</td>
<td>★★</td>
<td>0.5</td>
<td>980</td>
</tr>
</tbody>
</table>

select name, address
from Hotel
skyline of stars max,
dist min,
price min

- Currently considering
- "trumps" current
- Skyline
- Not skyline
Consider a **Hotel** table with columns **name**, **address**, **dist** (distance to the beach), **stars** (quality rating), & **price**.

```
select name, address
from Hotel
skyline of stars max,
dist min,
price min
```

```
<table>
<thead>
<tr>
<th>name</th>
<th>stars</th>
<th>dist</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aga</td>
<td>⭐⭐</td>
<td>0.7</td>
<td>1,175</td>
</tr>
<tr>
<td>Fol</td>
<td>⭐</td>
<td>1.2</td>
<td>1,237</td>
</tr>
<tr>
<td>Kaz</td>
<td>⭐</td>
<td>0.2</td>
<td>750</td>
</tr>
<tr>
<td>Neo</td>
<td>⭐⭐⭐</td>
<td>0.2</td>
<td>2,250</td>
</tr>
<tr>
<td>Tor</td>
<td>⭐⭐⭐</td>
<td>0.5</td>
<td>2,550</td>
</tr>
<tr>
<td>Uma</td>
<td>⭐⭐</td>
<td>0.5</td>
<td>980</td>
</tr>
</tbody>
</table>
```
A Skyline Example

Consider a **Hotel** table with columns **name**, **address**, **dist** (distance to the beach), **stars** (quality rating), & **price**.

<table>
<thead>
<tr>
<th>name</th>
<th>stars</th>
<th>dist</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aga</td>
<td>⭐⭐</td>
<td>0.7</td>
<td>1,175</td>
</tr>
<tr>
<td>Fol</td>
<td>⭐</td>
<td>1.2</td>
<td>1,237</td>
</tr>
<tr>
<td>Kaz</td>
<td>⭐</td>
<td>0.2</td>
<td>750</td>
</tr>
<tr>
<td>Neo</td>
<td>⭐⭐⭐</td>
<td>0.2</td>
<td>2,250</td>
</tr>
<tr>
<td>Tor</td>
<td>⭐⭐⭐</td>
<td>0.5</td>
<td>2,550</td>
</tr>
<tr>
<td>Uma</td>
<td>⭐⭐</td>
<td>0.5</td>
<td>980</td>
</tr>
</tbody>
</table>

| select name, address from Hotel skyline of stars max, dist min, price min |

- **Blue**: currently considering
- **Green**: “trumps” current
- **Red**: skyline
- **Gray**: not skyline
A Skyline Example

Consider a **Hotel** table with columns **name**, **address**, **dist** (distance to the beach), **stars** (quality rating), & **price**.

```sql
select name, address
from Hotel
skyline of stars max,
dist min,
price min
```

<table>
<thead>
<tr>
<th>name</th>
<th>stars</th>
<th>dist</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aga</td>
<td>★★★</td>
<td>0.7</td>
<td>1,175</td>
</tr>
<tr>
<td>Fol</td>
<td>★★★</td>
<td>1.2</td>
<td>1,237</td>
</tr>
<tr>
<td>Kaz</td>
<td>★★★</td>
<td>0.2</td>
<td>750</td>
</tr>
<tr>
<td>Neo</td>
<td>★★★★</td>
<td>0.2</td>
<td>2,250</td>
</tr>
<tr>
<td>Tor</td>
<td>★★★★</td>
<td>0.5</td>
<td>2,550</td>
</tr>
<tr>
<td>Uma</td>
<td>★★★</td>
<td>0.5</td>
<td>980</td>
</tr>
</tbody>
</table>

- Currently considering: Blue
- “trumps” current: Green
- Skyline: Red
- Not skyline: Gray
A Skyline Example

Consider a **Hotel** table with columns **name**, **address**, **dist** (distance to the beach), **stars** (quality rating), & **price**.

<table>
<thead>
<tr>
<th>name</th>
<th>stars</th>
<th>dist</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aga</td>
<td>⭐⭐</td>
<td>0.7</td>
<td>1,175</td>
</tr>
<tr>
<td>Fol</td>
<td>⭐</td>
<td>1.2</td>
<td>1,237</td>
</tr>
<tr>
<td>Kaz</td>
<td>⭐</td>
<td>0.2</td>
<td>750</td>
</tr>
<tr>
<td>Neo</td>
<td>⭐⭐⭐</td>
<td>0.2</td>
<td>2,250</td>
</tr>
<tr>
<td>Tor</td>
<td>⭐⭐⭐</td>
<td>0.5</td>
<td>2,550</td>
</tr>
<tr>
<td>Uma</td>
<td>⭐⭐</td>
<td>0.5</td>
<td>980</td>
</tr>
</tbody>
</table>

- **Aga** currently considering
- **Fol** “trumps” current
- **Kaz** skyline
- **Neo** not skyline
- **Tor** not skyline
- **Uma** not skyline
A Skyline Example

Consider a **Hotel** table with columns name, address, dist (distance to the beach), stars (quality rating), & price.

```
select name, address
from Hotel
skyline of stars max, dist min, price min
```

<table>
<thead>
<tr>
<th>name</th>
<th>stars</th>
<th>dist</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aga</td>
<td>⭐⭐</td>
<td>0.7</td>
<td>1,175</td>
</tr>
<tr>
<td>Fol</td>
<td>⭐</td>
<td>1.2</td>
<td>1,237</td>
</tr>
<tr>
<td>Kaz</td>
<td>⭐</td>
<td>0.2</td>
<td>750</td>
</tr>
<tr>
<td>Neo</td>
<td>⭐⭐⭐</td>
<td>0.2</td>
<td>2,250</td>
</tr>
<tr>
<td>Tor</td>
<td>⭐⭐⭐</td>
<td>0.5</td>
<td>2,550</td>
</tr>
<tr>
<td>Uma</td>
<td>⭐⭐</td>
<td>0.5</td>
<td>980</td>
</tr>
</tbody>
</table>

- **Currently considering**: Blue
- **“trumps” current**: Green
- **Skyline**: Red
- **Not skyline**: Gray

Maximal Vector—Godfrey, Shipley, & Gryz – p. 3/29
A Skyline Example

Consider a **Hotel** table with columns **name**, **address**, **dist** (distance to the beach), **stars** (quality rating), & **price**.

```
<table>
<thead>
<tr>
<th>name</th>
<th>stars</th>
<th>dist</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aga</td>
<td>⋆⋆</td>
<td>0.7</td>
<td>1,175</td>
</tr>
<tr>
<td>Fol</td>
<td>⋆</td>
<td>1.2</td>
<td>1,237</td>
</tr>
<tr>
<td>Kaz</td>
<td>⋆</td>
<td>0.2</td>
<td>750</td>
</tr>
<tr>
<td>Neo</td>
<td>⋆⋆⋆</td>
<td>0.2</td>
<td>2,250</td>
</tr>
<tr>
<td>Tor</td>
<td>⋆⋆⋆</td>
<td>0.5</td>
<td>2,550</td>
</tr>
<tr>
<td>Uma</td>
<td>⋆⋆</td>
<td>0.5</td>
<td>980</td>
</tr>
</tbody>
</table>
```

- **Blue**: currently considering
- **Green**: “trumps” current
- **Red**: skyline
- **Gray**: not skyline
A Skyline Example

Consider a Hotel table with columns name, address, dist (distance to the beach), stars (quality rating), & price.

```
<table>
<thead>
<tr>
<th>name</th>
<th>stars</th>
<th>dist</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aga</td>
<td>⭐⭐</td>
<td>0.7</td>
<td>1,175</td>
</tr>
<tr>
<td>Fol</td>
<td>⭐</td>
<td>1.2</td>
<td>1,237</td>
</tr>
<tr>
<td>Kaz</td>
<td>⭐</td>
<td>0.2</td>
<td>750</td>
</tr>
<tr>
<td>Neo</td>
<td>⭐⭐⭐</td>
<td>0.2</td>
<td>2,250</td>
</tr>
<tr>
<td>Tor</td>
<td>⭐⭐⭐</td>
<td>0.5</td>
<td>2,550</td>
</tr>
<tr>
<td>Uma</td>
<td>⭐⭐</td>
<td>0.5</td>
<td>980</td>
</tr>
</tbody>
</table>
```

select name, address from Hotel
skyline of stars max, dist min, price min

- currently considering
- “trumps” current
- skyline
- not skyline
A Skyline Example

Consider a **Hotel** table with columns **name**, **address**, **dist** (distance to the beach), **stars** (quality rating), & **price**.

```sql
select name, address
from Hotel
skyline of stars max,
  dist min,
  price min
```

<table>
<thead>
<tr>
<th>name</th>
<th>stars</th>
<th>dist</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aga</td>
<td>⭐⭐</td>
<td>0.7</td>
<td>1,175</td>
</tr>
<tr>
<td>Fol</td>
<td>⭐</td>
<td>1.2</td>
<td>1,237</td>
</tr>
<tr>
<td>Kaz</td>
<td>⭐</td>
<td>0.2</td>
<td>750</td>
</tr>
<tr>
<td>Neo</td>
<td>⭐⭐⭐</td>
<td>0.2</td>
<td>2,250</td>
</tr>
<tr>
<td>Tor</td>
<td>⭐⭐⭐</td>
<td>0.5</td>
<td>2,550</td>
</tr>
<tr>
<td>Uma</td>
<td>⭐⭐</td>
<td>0.5</td>
<td>980</td>
</tr>
</tbody>
</table>

- Blue: currently considering
- Green: “trumps” current
- Red: skyline
- Gray: not skyline

**Maximal Vector**—Godfrey, Shipley, & Gryz – p. 3/29
A Skyline Example

Consider a **Hotel** table with columns **name**, **address**, **dist** (distance to the beach), **stars** (quality rating), & **price**.

<table>
<thead>
<tr>
<th>name</th>
<th>stars</th>
<th>dist</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aga</td>
<td>⭐⭐</td>
<td>0.7</td>
<td>1,175</td>
</tr>
<tr>
<td>Fol</td>
<td>*</td>
<td>1.2</td>
<td>1,237</td>
</tr>
<tr>
<td>Kaz</td>
<td>*</td>
<td>0.2</td>
<td>750</td>
</tr>
<tr>
<td>Neo</td>
<td>⭐⭐⭐</td>
<td>0.2</td>
<td>2,250</td>
</tr>
<tr>
<td>Tor</td>
<td>⭐⭐⭐</td>
<td>0.5</td>
<td>2,550</td>
</tr>
<tr>
<td>Uma</td>
<td>⭐⭐</td>
<td>0.5</td>
<td>980</td>
</tr>
</tbody>
</table>

- currently considering
- “trumps” current
- skyline
- not skyline
A Skyline Example

Consider a **Hotel** table with columns **name**, **address**, **dist** (distance to the beach), **stars** (quality rating), & **price**.

```
select name, address
from Hotel
skyline of stars max,
dist min,
price min
```

<table>
<thead>
<tr>
<th>name</th>
<th>stars</th>
<th>dist</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aga</td>
<td>⭐⭐</td>
<td>0.7</td>
<td>1,175</td>
</tr>
<tr>
<td>Fol</td>
<td>⭐</td>
<td>1.2</td>
<td>1,237</td>
</tr>
<tr>
<td>Kaz</td>
<td>⭐</td>
<td>0.2</td>
<td>750</td>
</tr>
<tr>
<td>Neo</td>
<td>⭐⭐⭐</td>
<td>0.2</td>
<td>2,250</td>
</tr>
<tr>
<td>Tor</td>
<td>⭐⭐⭐</td>
<td>0.5</td>
<td>2,550</td>
</tr>
<tr>
<td>Uma</td>
<td>⭐⭐</td>
<td>0.5</td>
<td>980</td>
</tr>
</tbody>
</table>

*Blue* currently considering

*Green* “trumps” current

*Red* skyline

*Gray* not skyline
Consider a **Hotel** table with columns **name**, **address**, **dist** (distance to the beach), **stars** (quality rating), & **price**.

### Select name, address from Hotel skyline of stars max, dist min, price min

<table>
<thead>
<tr>
<th>name</th>
<th>stars</th>
<th>dist</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aga</td>
<td>⭐⭐</td>
<td>0.7</td>
<td>1,175</td>
</tr>
<tr>
<td>Fol</td>
<td>⭐</td>
<td>1.2</td>
<td>1,237</td>
</tr>
<tr>
<td>Kaz</td>
<td>⭐</td>
<td>0.2</td>
<td>750</td>
</tr>
<tr>
<td>Neo</td>
<td>⭐⭐⭐</td>
<td>0.2</td>
<td>2,250</td>
</tr>
<tr>
<td>Tor</td>
<td>⭐⭐⭐</td>
<td>0.5</td>
<td>2,550</td>
</tr>
<tr>
<td>Uma</td>
<td>⭐⭐</td>
<td>0.5</td>
<td>980</td>
</tr>
</tbody>
</table>

- **Currently considering**
- **“trumps” current**
- **Skyline**
- **Not skyline**
A Skyline Example

Consider a **Hotel** table with columns **name**, **address**, **dist** (distance to the beach), **stars** (quality rating), & **price**.

```sql
select name, address
from Hotel
skyline of stars max,
dist min,
price min
```

<table>
<thead>
<tr>
<th>name</th>
<th>stars</th>
<th>dist</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aga</td>
<td>⋆⋆</td>
<td>0.7</td>
<td>1,175</td>
</tr>
<tr>
<td>Fol</td>
<td>⋆</td>
<td>1.2</td>
<td>1,237</td>
</tr>
<tr>
<td>Kaz</td>
<td>⋆</td>
<td>0.2</td>
<td>750</td>
</tr>
<tr>
<td>Neo</td>
<td>⋆⋆⋆</td>
<td>0.2</td>
<td>2,250</td>
</tr>
<tr>
<td>Tor</td>
<td>⋆⋆⋆</td>
<td>0.5</td>
<td>2,550</td>
</tr>
<tr>
<td>Uma</td>
<td>⋆⋆</td>
<td>0.5</td>
<td>980</td>
</tr>
</tbody>
</table>
```

- Blue: currently considering
- Green: “trumps” current
- Red: skyline
- Gray: not skyline
The Maximal Vector Problem
Abstraction

Interest since the 1960’s.

tuples \approx \text{vectors (or points)}
in \(k\)-dim. space

Related to

- nearest neighbours
- convex hull

E.g., \(\langle \text{stars, dist, price} \rangle \rightarrow \langle x, y, z \rangle\)
The Maximal Vector Problem

Abstraction

Interest since the 1960’s.

tuples \approx \text{vectors (or points)}
\text{in } k\text{-dim. space}

Related to

- nearest neighbours
- convex hull

E.g., \langle \text{stars, dist, price} \rangle \mapsto \langle x, y, z \rangle

Input Set:

- \( n \) vectors
- \( k \) dimensions

Vectors (points) are scattered in the unit \( k \)-cube, \((0, 1)^k\).
The Maximal Vector Problem

Abstraction

Interest since the 1960’s.

tuples ≈ vectors (or points)
in $k$-dim. space

Related to

- nearest neighbours
- convex hull

E.g., $\langle\text{stars, dist, price}\rangle \mapsto \langle x, y, z\rangle$

**Input Set:**
- $n$ vectors
- $k$ dimensions

**Output Set:**
- $m$ maximal vectors

Vectors (points) are scattered in the unit $k$-cube, $(0, 1)^k$. 
1. To design a good relational-database algorithm for finding the maximal vectors / skyline: LESS

- performance criteria?
- design choices?
- computational issues?
Our Goals & Accomplishments

1. To design a good relational-database algorithm for finding the maximal vectors / skyline: LESS
   - performance criteria?
   - design choices?
   - computational issues?

2. To understand the strengths and weaknesses of the existing algorithms.
   - deeper asymptotic analyses
     *What is the impact of the dimensionality k?*
   - better analytic profiles
Our Goals & Accomplishments

1. To design a good relational-database algorithm for finding the maximal vectors / skyline: LESS
   - performance criteria?
   - design choices?
   - computational issues?

2. To understand the strengths and weaknesses of the existing algorithms.
   - deeper asymptotic analyses
     What is the impact of the dimensionality \( k \)?
   - better analytic profiles

We discuss #2 first.
II. Design & Analysis Considerations

Relational Performance Criteria

- **external**
  - I/O conscious (too much data for main memory)

- **well behaved**
  - compatible with a query optimizer
  - not CPU bound (!)

- **generic** *(At least one basic generic algorithm is needed!)*
  - no indexes, no pre-computed information.

- **good properties**
  - progressive, pipe-lineable
  - at worse, linear run-time (!)
<table>
<thead>
<tr>
<th><strong>Design Choices</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>divide-and-conquer (D&amp;C) or scan-based</strong></td>
</tr>
<tr>
<td>- Can D&amp;C be I/O conscious?</td>
</tr>
<tr>
<td>- Can scan-based be efficient?</td>
</tr>
<tr>
<td><strong>to sort or not to sort</strong></td>
</tr>
<tr>
<td>- Is sorting useful?</td>
</tr>
<tr>
<td>- Is sorting too inefficient? (Not linear...)</td>
</tr>
<tr>
<td><strong>comparison policy</strong></td>
</tr>
<tr>
<td>- Which vectors to compare next?</td>
</tr>
<tr>
<td>- How to limit the number of comparisons?</td>
</tr>
</tbody>
</table>
A Model for Average-Case Analysis

1. independence: Dimensions are statistically independent.
A Model for Average-Case Analysis

1. **independence**: Dimensions are statistically independent.

2. **sparseness**: Vectors (mostly) have distinct values along any dimension.
A Model for Average-Case Analysis

1. **independence**: Dimensions are statistically independent.

2. **sparseness**: Vectors (mostly) have distinct values along any dimension.

3. **uniformity**: The values along any dimension are uniformly distributed.
A Model for Average-Case Analysis

Component Independence (CI)

1. **independence**: Dimensions are statistically independent.

2. **sparseness**: Vectors (mostly) have distinct values along any dimension.

3. **uniformity**: The values along any dimension are uniformly distributed.
A Model for Average-Case Analysis

Uniform Independence (UI)

Component Independence (CI)

1. **independence**: Dimensions are statistically independent.

2. **sparseness**: Vectors (mostly) have distinct values along any dimension.

3. **uniformity**: The values along any dimension are uniformly distributed.
Expected Number of Maximals ($\hat{m}$)

Under CI (independence & sparseness),

$$\hat{m}_{1,n} = 1$$

$$\hat{m}_{k,n} = \frac{1}{n} \hat{m}_{k-1,n} + \hat{m}_{k,n-1}$$

[Bentley, Kung, Schkolnick, & Thompson 1978 (JACM)]

[Godfrey 2004 (FoIKS)]
**Expected Number of Maximals ($\hat{m}$)**

**Roman harmonics:**

\[
\begin{align*}
H_{0,n} &= 1 \\
H_{1,n} &= \sum_{i=1}^{n} \frac{1}{i} \\
H_{k,n} &= \sum_{i=1}^{n} \frac{H_{k-1,i}}{i} \\
H_{k,n} &\approx \frac{1}{k!} \ln(kn)
\end{align*}
\]

**Under CI (independence & sparseness),**

\[
\begin{align*}
\hat{m}_{1,n} &= 1 \\
\hat{m}_{k,n} &= \frac{1}{n} \hat{m}_{k-1,n} + \hat{m}_{k,n-1}
\end{align*}
\]

[Bentley, Kung, Schkolnick, & Thompson 1978 (JACM)]

[Godfrey 2004 (FoIKS)]

[Roman 2004 (AMM)]
Expected Number of Maximals ($\hat{m}$)

Roman harmonics:

\[
\begin{align*}
H_{0,n} &= 1 \\
H_{1,n} &= \sum_{i=1}^{n} \frac{1}{i} \\
H_{k,n} &= \sum_{i=1}^{n} \frac{H_{k-1,i}}{i} \\
H_{k,n} &\approx \frac{1}{k!} \ln k \, n
\end{align*}
\]

Under CI (independence & sparseness),

\[
\begin{align*}
\hat{m}_{1,n} &= 1 \\
\hat{m}_{k,n} &= \frac{1}{n} \hat{m}_{k-1,n} + \hat{m}_{k,n-1}
\end{align*}
\]

\[
\hat{m}_{k,n} = H_{k-1,n}
\]

[Bentley, Kung, Schkolnick, & Thompson 1978 (JACM)]
[Godfrey 2004 (FoIKS)]
[Roman 2004 (AMM)]
III. Algorithms & Analyses

Existing Generic Algorithms

- **Divide-and-Conquer Algorithms**
  - **DD&C**: double divide and conquer [Kung, Luccio, & Preparata 1975 (JACM)]
  - **LD&C**: linear divide and conquer [Bentley, Kung, Schkolnick, & Thompson 1978 (JACM)]
  - **FLET**: fast linear expected time [Bentley, Clarkson, & Levine 1990 (SODA)]
  - **SD&C**: single divide and conquer [Börzsönyi, Kossmann, & Stocker 2001 (ICDE)]

- **Scan-based (Relational “Skyline”) Algorithms**
  - **BNL**: block nested loops [Börzsönyi, Kossmann, & Stocker 2001 (ICDE)]
  - **SFS**: sort filter skyline [Chomicki, Godfrey, Gryz, & Liang 2003 (ICDE)]
  - **LESS**: linear elimination sort for skyline [Godfrey, Shipley, & Gryz 2005 (VLDB)]
D&C: Comparisons per Vector

We know $\hat{m}$ (under CI), so we can model \textit{and} solve a recurrence relation that is a floor for a D&C algorithm’s average-case in terms of $n$ \textit{and} $k$. \textbf{LD&C [BKST 1978 (JACM)]}:
We know $\hat{m}$ (under CI), so we can model \textit{and} solve a recurrence relation that is a floor for a D&C algorithm’s average-case in terms of $n$ \textit{and} $k$. \textbf{LD&C} [BKST 1978 (JACM)]:

\[
T(n) = 2T(n/2) + \hat{m}_{k,n}\lfloor \log_2 k \rfloor \hat{m}_{k,n}
\]
\[
\vdots
\]
\[
\approx (k - 1)^{k-2} n
\]
D&C: Comparisons per Vector

We know \( \hat{m} \) (under CI), so we can model and solve a recurrence relation that is a floor for a D&C algorithm’s average-case in terms of \( n \) and \( k \). LD&C [BKST 1978 (JACM)]:

\[
T(n) = 2T(n/2) + \hat{m}_{k,n} \log_2^{k-2} \hat{m}_{k,n} \\
\vdots \\
\approx (k - 1)^{k-2} n
\]

<table>
<thead>
<tr>
<th>( k )</th>
<th>( (k - 1)^{k-2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>64</td>
</tr>
<tr>
<td>7</td>
<td>7,776</td>
</tr>
<tr>
<td>9</td>
<td>2,097,152</td>
</tr>
</tbody>
</table>
D&C: Comparisons per Vector

We know $\hat{m}$ (under CI), so we can model and solve a recurrence relation that is a floor for a D&C algorithm’s average-case in terms of $n$ and $k$. **LD&C** [BKST 1978 (JACM)]:

$$T(n) = 2T(n/2) + \hat{m}_{k,n}\log_2^{k-2}\hat{m}_{k,n}$$

$$\approx (k - 1)^{k-2}n$$

<table>
<thead>
<tr>
<th>$k$</th>
<th>$(k - 1)^{k-2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>64</td>
</tr>
<tr>
<td>7</td>
<td>7,776</td>
</tr>
<tr>
<td>9</td>
<td>2,097,152</td>
</tr>
</tbody>
</table>
We know $\hat{m}$ (under CI), so we can model and solve a recurrence relation that is a floor for a D&C algorithm’s average-case in terms of $n$ and $k$. **LD&C [BKST 1978 (JACM)]:**

$$T(n) = 2T(n/2) + \hat{m}_{k,n} \lg \frac{k-2}{2} \hat{m}_{k,n}$$

$$\approx (k - 1)^{k-2} n$$

<table>
<thead>
<tr>
<th>$k$</th>
<th>$(k - 1)^{k-2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>64</td>
</tr>
<tr>
<td>7</td>
<td>7,776</td>
</tr>
<tr>
<td>9</td>
<td>2,097,152</td>
</tr>
</tbody>
</table>

**DD&C [KLP 1975 (JACM)]:**

$$(k - 1)^{k-3} n$$

**SD&C [BKS 2001 (ICDE)]:**

$$\frac{\ln 2}{\sqrt{\pi(k-1)}} 2^{2k-4} n$$
**Block Nested Loops (BNL) Algorithm**

`window (W)`: A fixed size of main memory used to store skyline-candidate vectors (tuples).

`stream (S)`: The \( n \) vectors (tuples) resident on disk, to be read in “one-by-one”.

\[
\begin{align*}
\text{for each } \vec{v} & \in S \\
\text{for each } \vec{w} & \in W \\
\text{if } (\vec{w} \triangleright \vec{v}) \\
\text{continue} & \quad // \text{with next } \vec{v} \\
\text{if } (\vec{v} \triangleright \vec{w}) \\
W & := W - \{\vec{w}\} \\
\text{if } (\neg \exists \vec{w} \in W. \vec{w} \triangleright \vec{v}) & \quad // \vec{v} \text{ survived} \\
W & := W \cup \{\vec{v}\} & \quad // \text{if there is room}
\end{align*}
\]

\( O(?) \)

average case
### Sort Filter Skyline (SFS) Algorithm

Have a **window** ($W$) and **stream** ($S$), as with BNL. Sort $S$ first (via an external sort routine): e.g.,

<table>
<thead>
<tr>
<th>order by $D_k$ desc, $\ldots$, $D_1$ desc</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{O}(n \log n)$ worst case</td>
</tr>
</tbody>
</table>

Then,

<table>
<thead>
<tr>
<th>for each $\vec{v} \in S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>for each $\vec{w} \in W$</td>
</tr>
<tr>
<td>if ($\vec{w} \succ \vec{v}$)</td>
</tr>
<tr>
<td>continue // with next $\vec{v}$</td>
</tr>
<tr>
<td>if ($\vec{v} \succ \vec{w}$)</td>
</tr>
<tr>
<td>$W := W {\vec{w}}$</td>
</tr>
<tr>
<td>if ($\not\exists \vec{w} \in W. \vec{w} \succ \vec{v}$) // $\vec{v}$ survived</td>
</tr>
<tr>
<td>$W := W \cup {\vec{v}}$ // if there is room</td>
</tr>
</tbody>
</table>

$\mathcal{O}(n)$ average case

Thm. 8 (under UI & sort on entropy)

Any $\vec{w}$ in the window is guaranteed to be maximal (skyline).
BNL vs SFS

- SFS makes fewer comparisons and takes fewer passes.
- SFS is better behaved “relationally”.
  - progressive
  - immune to previous ordering of input
- BNL does not need to sort!
  (However, what is its average-case $O$?)
BNL vs SFS

- SFS makes fewer comparisons and takes fewer passes.
- SFS is better behaved “relationally”.
  - progressive
  - immune to previous ordering of input
- BNL does not need to sort!
  (However, what is its average-case $O$?)

Our algorithm LESS will combine the best aspects of the algorithms, particularly of BNL & SFS.
BNL vs SFS

- SFS makes fewer comparisons and takes fewer passes.
- SFS is better behaved “relationally”.
  - progressive
  - immune to previous ordering of input
- BNL does not need to sort!
  (However, what is its average-case $O$?)

$\text{BNL}_R \& \text{SFS}_R$: Compare $\vec{v}$ against window $\vec{w}$’s in a random order.

$\text{BNL} \& \text{SFS}$: Order window $\vec{w}$’s intelligently to reduce #comparisons.
Analyses of #Comparisons

BNL$_R$:

$$\sum_{i=0}^{n-1} \int_{x_k=0}^{1} \int_{x_{k-1}=0}^{1} \ldots \int_{x_1=0}^{1} \hat{\text{mttf}}_k(x_1 \cdot \ldots \cdot x_k, i) \, dx_1 \ldots dx_k$$
BNL$_R^i$:

\[
\sum_{i=0}^{n-1} \int_{x_k=0}^{1} \int_{x_{k-1}=0}^{1} \cdots \int_{x_1=0}^{1} \hat{mttf}_k(x_1, \ldots, x_k, i) \, dx_1 \ldots dx_k
\]

\textit{mttf}: “mean time to failure”
BNL\(_R\):

\[ \sum_{i=0}^{n-1} \int_{x_k=0}^{1} \int_{x_{k-1}=0}^{1} \cdots \int_{x_1=0}^{1} \widehat{\text{mttf}}_k(x_1 \cdot \ldots \cdot x_k, i) \, dx_1 \cdots dx_k \]

\text{mttf}: "mean time to failure"
BNL_R:

\[ \sum_{i=0}^{n-1} \int_{x_k=0}^{1} \int_{x_{k-1}=0}^{1} \cdots \int_{x_1=0}^{1} \hat{\text{mttf}}_k(x_1, \ldots, x_k, i) \, dx_1 \cdots dx_k \]
BNL_R:\n\int_{z=0}^{1} \int_{x_k=0}^{1} \int_{x_{k-1}=0}^{1} \cdots \int_{x_1=0}^{1} \hat{\text{mttf}}_k(x_1, \ldots, x_k, z, n) \, dx_1 \cdots dx_k \, dz
Analyses of #Comparisons

**BNL** \(_R\): 
\[ \int_{z=0}^{1} \int_{x_k=0}^{1} \int_{x_{k-1}=0}^{1} \ldots \int_{x_1=0}^{1} \widehat{\text{mttf}}_k(x_1 \ldots x_k, zn) \, dx_1 \ldots dx_k \, dz \]

**SFS** \(_R\) w/o elimination from window: 
\[ \int_{z=0}^{1} \int_{x_{k-1}=0}^{1} \ldots \int_{x_1=0}^{1} \widehat{\text{mttf}}_k(x_1 \ldots x_{k-1}, zn) \, dx_1 \ldots dx_{k-1} \, dz \]
Analyses of #Comparisons

BNL<sub>R</sub>:  
\[ \int_{z=0}^{1} \int_{x_k=0}^{1} \int_{x_{k-1}=0}^{1} \ldots \int_{x_1=0}^{1} \hat{\text{mttf}}_k(x_1 \ldots \cdot x_k, z_n) \, dx_1 \ldots dx_k \, dz \]

SFS<sub>R</sub> w/o elimination from window:  
\[ \int_{z=0}^{1} \int_{x_{k-1}=0}^{1} \ldots \int_{x_1=0}^{1} \hat{\text{mttf}}_k(x_1 \ldots \cdot x_{k-1}, z_n) \, dx_1 \ldots dx_{k-1} \, dz \]

SFS<sub>R</sub> w/ elimination from window:  
\[ \int_{z=0}^{1} \int_{x_{k-1}=0}^{1} \ldots \int_{x_1=0}^{1} \hat{\text{mttf}}_{k-1}(x_1 \ldots \cdot x_{k-1}, z_n) \, dx_1 \ldots dx_{k-1} \, dz \]
Analyses of #Comparisons

$\text{BNL}_R$: 
\[ \int_{z=0}^{1} \int_{x_k=0}^{1} \int_{x_{k-1}=0}^{1} \ldots \int_{x_1=0}^{1} \mathcal{mttf}_k(x_1 \cdot \ldots \cdot x_k, z, n) \, dx_1 \ldots dx_k \, dz \]

$\text{SFS}_R$ w/o elimination from window: 
\[ \int_{z=0}^{1} \int_{x_{k-1}=0}^{1} \ldots \int_{x_1=0}^{1} \mathcal{mttf}_k(x_1 \cdot \ldots \cdot x_{k-1}, z, n) \, dx_1 \ldots dx_{k-1} \, dz \]

$\text{SFS}_R$ w/ elimination from window: 
\[ \int_{z=0}^{1} \int_{x_{k-1}=0}^{1} \ldots \int_{x_1=0}^{1} \mathcal{mttf}_{k-1}(x_1 \cdot \ldots \cdot x_{k-1}, z, n) \, dx_1 \ldots dx_{k-1} \, dz \]

$\text{SFS}$ effectively saves “one dimension” over $\text{BNL}$. 

Maximal Vector—Godfrey, Shipley, & Gryz – p. 15/29
Analyses of #Comparisons

Results

\[ \hat{\text{mttf}}_k(x, n) \approx \frac{H_{k-1,n}}{H_{k-1, xn}} \]

These converge in the limit.
Analyses of #Comparisons

Results

\[ \text{mttf}_k(x, n) \approx \frac{H_{k-1,n}}{H_{k-1,xn}} \]

These converge in the limit.

Analytical solution matches observation.
Analyses of #Comparisons

Results

\[ \widehat{\text{mttf}}_k(x, n) \approx \frac{H_{k-1,n}}{H_{k-1,xn}} \]

These converge in the limit.

Analytical solution matches observation.

**Thm.** Under CI, BNL\(_R\) and SFS\(_R\) are \(O(n)\) average case.

**Proof.**

\[
\lim_{n \to \infty} \int_{z=0}^{1} \int_{x_k=0}^{1} \ldots \int_{x_1=0}^{1} \widehat{\text{mttf}}_k(\ldots, zn) \, d \ldots = 1
\]
BNL & SFS

Comparisons per Vector

#comparisons per vector

#vectors

BNL $R$

SFS $R$ w/o

SFS $R$ w/

$k = 7$
BNL & SFS

Comparisons per Vector

#comparisons per vector

#vectors

BNL $R$
BNF
SFS $R$ w/o
SFS $R$ w/
SFS

$k = 7$

Maximal Vector—Godfrey, Shipley, & Gryz – p. 17/29
The LESS Algorithm

Description

Combine best aspects of the algorithms, mainly BNL & SFS.

- modified external sort
- block-sort pass
  - use a small window (as in BNL) to eliminate $\vec{v}$'s
  - merge passes
    - ...
  - last merge pass
    - use a large window (as in SFS) to filter for the skyline
    - skyline-filter passes (if needed)
    - ...

Buffer Pool

EF Window

Block for quicksort

Last merge pass

buffer pool

SF Window

Inputs
Output
1
2
k

Maximal Vector—Godfrey, Shipley, & Gryz – p. 18/29
LESS: Performance

\[ n = 500,000 \]
EF window: 200 vectors
SF window: 76 pages, \( \sim 3,000 \) vectors
Pentium III, 733 MHz
RedHat Linux 7.3
LESS: Linear Average-Case

Summary

$O(n)$ average-case run-time (under UI, Thm. 13)

- BNL-style filtering during the block-sort pass removes enough so sort is $O(n)$.
- SFS-style filtering during the last merge pass (and subsequent filter-skyline passes) is $O(n)$.

Improvements

- LESS improves over SFS & BNL on I/O’s.
- LESS improves over SFS & BNL on time; however, for larger $k’s$ (and, hence, $m’s$), this diminishes.
Conclusions

Future Work

1. Devise yet better (generic) algorithms.
   - A scan-based algorithm that is $o(n^2)$ worst-case?
   - Can we bypass the $m^2$ bottleneck?
   - Make “average-case” more general.
     - Nemesis of skyline: anti-correlation.
     - Remove uniformity assumption.
   - Reduce further comparison load (CPU-boundness).

2. Study in depth index-based skyline algorithms.
   - What are their asymptotic complexities?
   - In what cases will a given index-based algorithm outperform, say, LESS? Not outperform?
1. Asymptotic complexity does not tell all. If you dig a little deeper, you often find surprises!
   - The multiplicative constant matters.
   - Even when the multiplicative constant is good in the limit, what happens in between?
   - Must factor in “database” considerations.

2. Maximal-vector / skyline opens up new & useful avenues for database systems.
   - Adds a preference facility to the language.
   - Provides a multi-objective operation.
   - May be useful in other applications.
Appendix

Extra Slides
Computing Skyline in (Plain) SQL

\[
\begin{align*}
\text{select } & C_1, \ldots, C_j, \quad \text{– columns to keep} \\
& D_1, \ldots, D_k, \quad \text{– skyline dimensions (MAX assumed)} \\
& E_1, \ldots, E_l \quad \text{– DIFF columns} \\
\text{from OurTable} \\
\text{except} \\
\text{select } & X.C_1, \ldots, X.C_j, \\
& X.D_1, \ldots, X.D_k, \\
& X.E_1, \ldots, X.E_l \\
\text{from OurTable } X, \text{ OurTable } Y \text{ where} \\
& Y.D_1 \geq X.D_1 \text{ and } \ldots \ Y.D_k \geq X.D_k \text{ and} \\
& (Y.D_1 > X.D_1 \text{ or } \ldots \ Y.D_k > X.D_k) \text{ and} \\
& Y.E_1 = X.E_1 \text{ and } \ldots \ Y.E_l = X.E_l
\end{align*}
\]

Certainly $O(n^2)$, even for average-case.
Skyline Cardinality

harmonic numbers [Godfrey 2004 (FoIKS)]

1. The harmonic of \( n \), for \( n > 0 \): \( H_n = \sum_{i=1}^{n} \frac{1}{i} \)

2. The \( k \)-th order harmonic of \( n \), for integers \( k > 0 \) and integers \( n > 0 \): \( H_{k,n} = \sum_{i=1}^{n} \frac{H_{k-1,i}}{i} \)

Define \( H_{0,n} = 1 \), for \( n > 0 \). Define \( H_{k,0} = 0 \), for \( k > 0 \).

3. The \( k \)-th hyper-harmonic of \( n \), for integers \( k > 0 \) and integers \( n > 0 \): \( H_{k,n} = \sum_{i=1}^{n} \frac{1}{i^k} \)

\[
\hat{m}_{k+1,n} = H_{k,n} = \sum_{i_{1}=1}^{n} \sum_{i_{2}=1}^{i_{1}} \cdots \sum_{i_{k}=1}^{i_{k-1}} \frac{1}{i_{1}i_{2}\cdots i_{k}}
\]
Thm.

\[ H_{k,n} = \sum_{c_1,\ldots,c_k \geq 0} \prod_{i=1}^{k} \frac{H_{ci}^{ci}}{i^{ci} \cdot c_i!} \]

for \( k \geq 1 \) and \( n \geq 1 \), with the \( c_i \)'s as integers.

Follows from Knuth’s generalization via generating functions.

- Only \( H_{1,n} (= H_n) \) diverges with \( n \).
- Each \( H_{i,n} \) for \( i > 1 \) converges.
- Thm. \( H_{k,n} \) is \( \Theta((\ln n)^k / k!) \).
- Thm. \( \hat{m}_{k,n} \) is \( \Theta((\ln n)^{k-1} / (k - 1)!) \).
Skyline Cardinality

examples [Godfrey 2004 (FoIKS)]

- \( H_{2,n} = \frac{1}{2} H_n^2 + \frac{1}{2} \mathcal{H}_{2,n} \),
- \( H_{3,n} = \frac{1}{6} H_n^3 + \frac{1}{2} H_n \mathcal{H}_{2,n} + \frac{1}{3} \mathcal{H}_{3,n} \), and
- \( H_{4,n} = \frac{1}{24} H_n^4 + \frac{1}{3} H_n \mathcal{H}_{3,n} + \frac{1}{8} \mathcal{H}_{2,n}^2 + \frac{1}{4} H_n^2 \mathcal{H}_{2,n} + \frac{1}{4} \mathcal{H}_{4,n} \).
- \( \ldots \)
D&C \mid +\text{Sort}

DD&C

1. Sort input set initially on each dimension.
2. Recursively divide (sorted) input set (along one dimension).
3. On merge, recursively call DD&C, but with one dimension fewer.

worst-case: $O(n\lg^{k−2} n)$

theoreticians: Great! $o(n^2)$!

engineers: Awful! $\lg^{k−2} n$ can be pretty large!

And, of course, average case is $\Omega(kn\lg n)$, because we have to sort.
D&C | –Sort

(Do not sort initially.)
1. Recursively divide input set.
2. On merge, call DD&C.

worst-case: $\mathcal{O}(n \lg^{k-1} n)$. Still $o(n^2)$!
average-case: $\mathcal{O}(n)$. Linear!
(Do not sort initially.)

1. Recursively divide input set.
2. On merge, call DD&C.

worst-case: $\mathcal{O}(n \lg^{k-1} n)$. Still $o(n^2)$!
average-case: $\mathcal{O}(n)$. Linear!

• So, is this a good algorithm?
• What is the “multiplicative constant”?
  – What impact does $k$ have?
  – How many comparisons per vector (#CpV) are needed, on average?