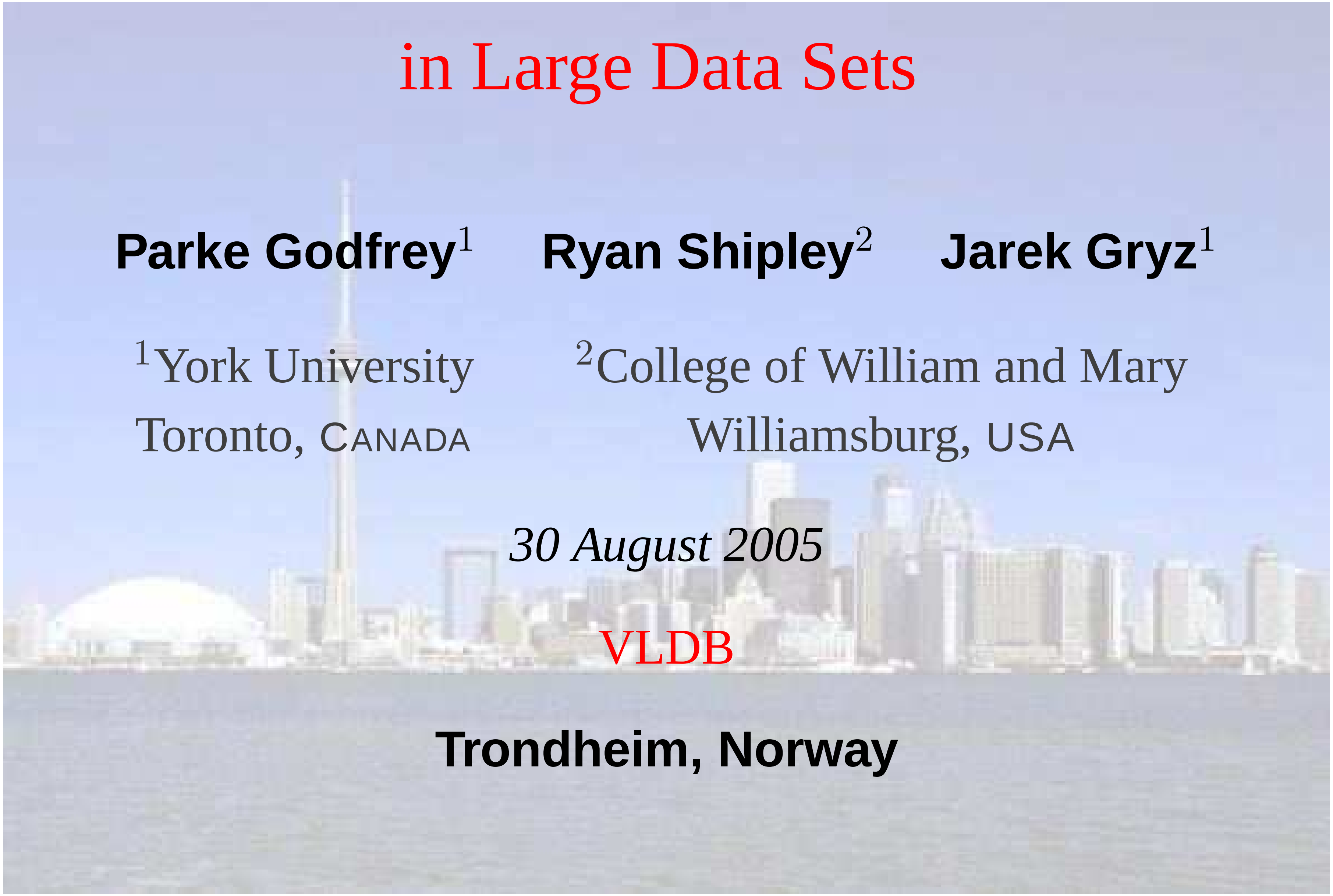
Maximal Vector Computation



in Large Data Sets

Parke Godfrey[1] **Ryan Shipley**[2] **Jarek Gryz**[1]

[1]York University
Toronto, CANADA

[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.

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
```

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
```

name	stars	dist	price
Aga	⋆⋆	0.7	1,175
Fol	⋆	1.2	1,237
Kaz	⋆	0.2	750
Neo	⋆⋆⋆	0.2	2,250
Tor	⋆⋆⋆	0.5	2,550
Uma	⋆⋆	0.5	980

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
```

- currently considering (blue)
- "trumps" current (green)
- skyline (red)
- not skyline (grey)

name	stars	dist	price
Aga	⋆⋆	0.7	1,175
Fol	⋆	1.2	1,237
Kaz	⋆	0.2	750
Neo	⋆⋆⋆	0.2	2,250
Tor	⋆⋆⋆	0.5	2,550
Uma	⋆⋆	0.5	980

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
```

name	stars	dist	price
Aga	⋆⋆	0.7	1,175
Fol	⋆	1.2	1,237
Kaz	⋆	0.2	750
Neo	⋆⋆⋆	0.2	2,250
Tor	⋆⋆⋆	0.5	2,550
Uma	⋆⋆	0.5	980

- currently considering (blue)
- "trumps" current (green)
- skyline (red)
- not skyline (grey)

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
```

name	stars	dist	price
Aga	⋆⋆	0.7	1,175
Fol	⋆	1.2	1,237
Kaz	⋆	0.2	750
Neo	⋆⋆⋆	0.2	2,250
Tor	⋆⋆⋆	0.5	2,550
Uma	⋆⋆	0.5	980

- 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
```

name	stars	dist	price
Aga	$\star\star$	0.7	1,175
Fol	$\star$	1.2	1,237
Kaz	$\star$	0.2	750
Neo	$\star\star\star$	0.2	2,250
Tor	$\star\star\star$	0.5	2,550
Uma	$\star\star$	0.5	980

- 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
```

name	stars	dist	price
Aga	⋆⋆	0.7	1,175
Fol	⋆	1.2	1,237
Kaz	⋆	0.2	750
Neo	⋆⋆⋆	0.2	2,250
Tor	⋆⋆⋆	0.5	2,550
Uma	⋆⋆	0.5	980

- 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
```

name	stars	dist	price
Aga	⋆⋆	0.7	1,175
Fol	⋆	1.2	1,237
Kaz	⋆	0.2	750
Neo	⋆⋆⋆	0.2	2,250
Tor	⋆⋆⋆	0.5	2,550
Uma	⋆⋆	0.5	980

- 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
```

- currently considering
- "trumps" current
- skyline
- not skyline

name	stars	dist	price
Aga	⋆⋆	0.7	1,175
Fol	⋆	1.2	1,237
Kaz	⋆	0.2	750
Neo	⋆⋆⋆	0.2	2,250
Tor	⋆⋆⋆	0.5	2,550
Uma	⋆⋆	0.5	980

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
```

name	stars	dist	price
Aga	⋆⋆	0.7	1,175
Fol	⋆	1.2	1,237
Kaz	⋆	0.2	750
Neo	⋆⋆⋆	0.2	2,250
Tor	⋆⋆⋆	0.5	2,550
Uma	⋆⋆	0.5	980

- 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
```

- currently considering (blue)
- "trumps" current (green)
- skyline (red)
- not skyline (grey)

name	stars	dist	price
Aga	⋆⋆	0.7	1,175
Fol	⋆	1.2	1,237
Kaz	⋆	0.2	750
Neo	⋆⋆⋆	0.2	2,250
Tor	⋆⋆⋆	0.5	2,550
Uma	⋆⋆	0.5	980

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
```

- currently considering
- “trumps” current
- skyline
- not skyline

name	stars	dist	price
Aga	⋆⋆	0.7	1,175
Fol	⋆	1.2	1,237
Kaz	⋆	0.2	750
Neo	⋆⋆⋆	0.2	2,250
Tor	⋆⋆⋆	0.5	2,550
Uma	⋆⋆	0.5	980

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
```

name	stars	dist	price
Aga	⋆⋆	0.7	1,175
Fol	⋆	1.2	1,237
Kaz	⋆	0.2	750
Neo	⋆⋆⋆	0.2	2,250
Tor	⋆⋆⋆	0.5	2,550
Uma	⋆⋆	0.5	980

- (blue) currently considering
- (green) “trumps” current
- (red) skyline
- (grey) 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
```

- currently considering
- "trumps" current
- skyline
- not skyline

name	stars	dist	price
Aga	⋆⋆	0.7	1,175
Fol	⋆	1.2	1,237
Kaz	⋆	0.2	750
Neo	⋆⋆⋆	0.2	2,250
Tor	⋆⋆⋆	0.5	2,550
Uma	⋆⋆	0.5	980

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
```

- currently considering
- "trumps" current
- skyline
- not skyline

name	stars	dist	price
Aga	⋆⋆	0.7	1,175
Fol	⋆	1.2	1,237
Kaz	⋆	0.2	750
Neo	⋆⋆⋆	0.2	2,250
Tor	⋆⋆⋆	0.5	2,550
Uma	⋆⋆	0.5	980

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
```

- currently considering
- “trumps” current
- skyline
- not skyline

name	stars	dist	price
Aga	⋆⋆	0.7	1,175
Fol	⋆	1.2	1,237
Kaz	⋆	0.2	750
Neo	⋆⋆⋆	0.2	2,250
Tor	⋆⋆⋆	0.5	2,550
Uma	⋆⋆	0.5	980

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
```

- currently considering
- "trumps" current
- skyline
- not skyline

name	stars	dist	price
Aga	⋆⋆	0.7	1,175
Fol	⋆	1.2	1,237
Kaz	⋆	0.2	750
Neo	⋆⋆⋆	0.2	2,250
Tor	⋆⋆⋆	0.5	2,550
Uma	⋆⋆	0.5	980

The Maximal Vector Problem

Abstraction

Interest since the 1960's.

tuples $\approx$ vectors (or points)
in *k*-dim. space

Related to

- nearest neighbours
- convex hull

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

The Maximal Vector Problem

Abstraction

Interest since the 1960's.

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

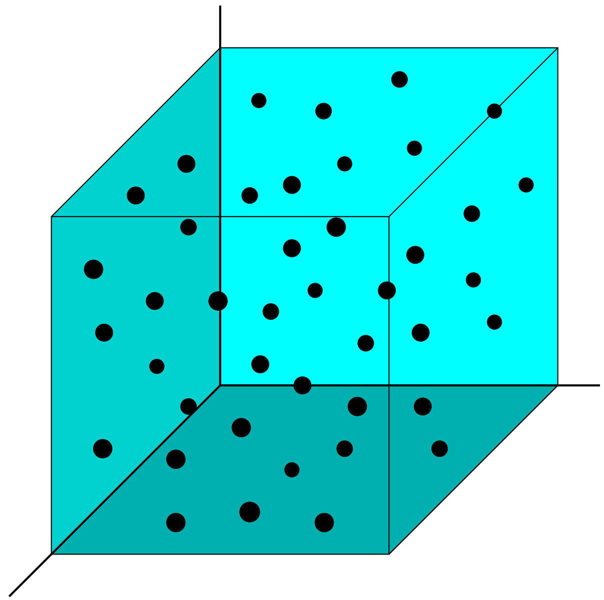
Related to

- nearest neighbours
- convex hull

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

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

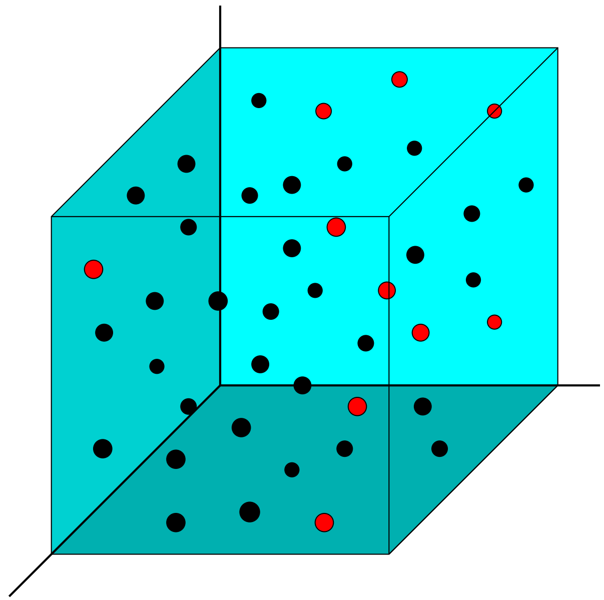
E.g., $\langle \text{stars}, \text{dist}, \text{price} \rangle \mapsto \langle x, y, z \rangle$

Input Set:

- n vectors
- k dimensions

Output Set:

- m maximal vectors



Vectors (points) are scattered in the unit k-cube, $(0, 1)^k$.

Our Goals & Accomplishments

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 (!)

Design Choices

- **divide-and-conquer (D&C) *or* scan-based**
  - Can D&C be I/O conscious?
  - Can scan-based be efficient?
- **to sort *or* not to sort**
  - Is sorting useful?
  - Is sorting too inefficient? (Not linear. . .)
- **comparison policy**
  - Which vectors to compare next?
  - How to limit the number of comparisons?

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 ($\widehat{m}$)

Under CI (independence & sparseness),

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

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

[Bentley, Kung, Schkolnick, & Thompson 1978 (JACM)]
[Godfrey 2004 (FoIKS)]

Expected Number of Maximals ($\widehat{m}$)

Roman harmonics:

$$\mathsf{H}_{0,n} = 1$$

$$\mathsf{H}_{1,n} = \sum_{i=1}^{n} \frac{1}{i}$$

$$\mathsf{H}_{k,n} = \sum_{i=1}^{n} \frac{\mathsf{H}_{k-1,i}}{i}$$

$$\mathsf{H}_{k,n} \approx \frac{1}{k!} \ln^k n$$

Under CI (independence & sparseness),

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

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

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

Expected Number of Maximals ($\widehat{m}$)

Roman harmonics:

$$\mathsf{H}_{0,n} = 1$$

$$\mathsf{H}_{1,n} = \sum_{i=1}^{n} \frac{1}{i}$$

$$\mathsf{H}_{k,n} = \sum_{i=1}^{n} \frac{\mathsf{H}_{k-1,i}}{i}$$

$$\mathsf{H}_{k,n} \approx \frac{1}{k!} \ln^k n$$

Under CI (independence & sparseness),

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

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

$$\widehat{m}_{k,n} = \mathsf{H}_{k-1,n}$$

[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)]
    [Chomicki, Godfrey, Gryz, & Liang 2005 (IIS)]
  - **LESS**: *linear elimination sort for skyline* [Godfrey, Shipley, & Gryz 2005 (VLDB)]

D&C: Comparisons per Vector

We know $\widehat{m}$ (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)]:

D&C: Comparisons per Vector

We know $\widehat{m}$ (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)]:

$$
\begin{aligned}
T(n) &= 2T(n/2) \\
&\quad + \widehat{m}_{k,n} \lg_2^{k-2} \widehat{m}_{k,n} \\
&\quad \vdots \\
&\approx (k-1)^{k-2} n
\end{aligned}
$$

D&C: Comparisons per Vector

We know $\widehat{m}$ (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)]:

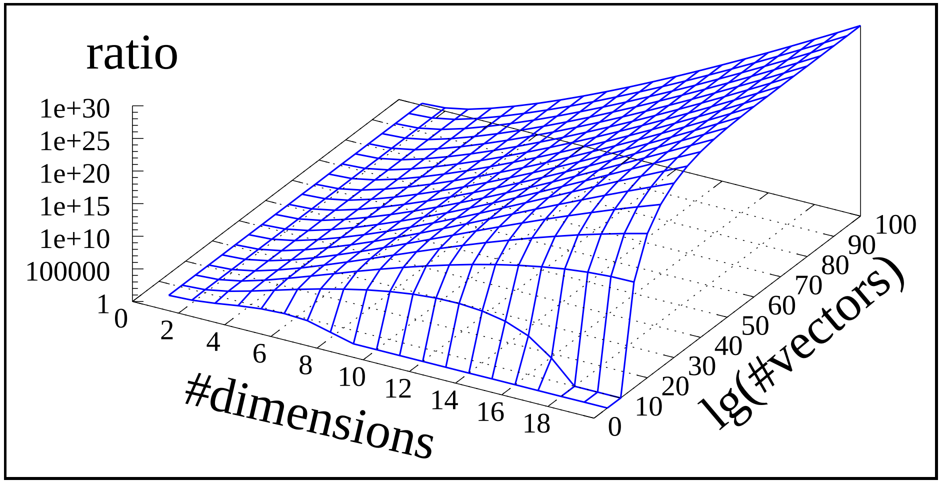
$$\begin{aligned} T(n) &= 2T(n/2) \\ &\quad + \widehat{m}_{k,n} \lg_2^{k-2} \widehat{m}_{k,n} \\ &\quad \vdots \\ &\approx (k-1)^{k-2} n \end{aligned}$$

k	$(k-1)^{k-2}$
5	64
7	7,776
9	2,097,152

D&C: Comparisons per Vector

We know $\widehat{m}$ (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)]:

$$\begin{aligned} T(n) &= 2T(n/2) \\ &\quad + \widehat{m}_{k,n} \lg_2^{k-2} \widehat{m}_{k,n} \\ &\quad \vdots \\ &\approx (k-1)^{k-2} n \end{aligned}$$

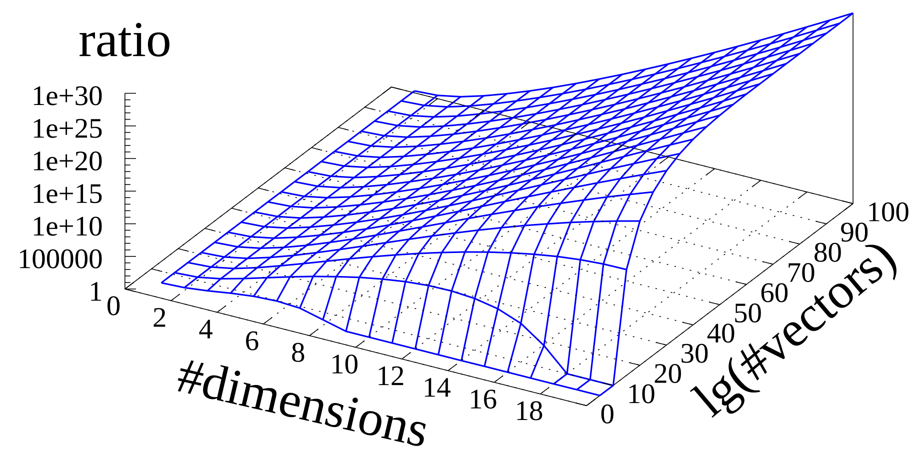


k	$(k-1)^{k-2}$
5	64
7	7,776
9	2,097,152

D&C: Comparisons per Vector

We know $\widehat{m}$ (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)]:

$$T(n) = 2T(n/2) + \widehat{m}_{k,n} \lg_2^{k-2} \widehat{m}_{k,n}$$
$$\vdots$$
$$\approx (k-1)^{k-2} n$$



k	$(k-1)^{k-2}$
5	64
7	7,776
9	2,097,152

- 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".

```
for each v⃗ ∈ S
    for each w⃗ ∈ W
        if (w⃗ ≻ v⃗)
            continue            // with next v⃗
        if (v⃗ ≻ w⃗)
            W := W − {w⃗}
    if (¬∃w⃗ ∈ W. w⃗ ≻ v⃗)       // v⃗ survived
        W := W ∪ {v⃗}            // if there is room
```

$\mathcal{O}(?)$
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.,

```
order by D_k desc, . . ., D_1 desc
```

$\mathcal{O}(n \lg n)$
worst case

Then,

for each $\vec{v} \in$ S
for each $\vec{w} \in$ W
if $(\vec{w} \succ \vec{v})$
continue   *// with next $\vec{v}$*
~~if $(\vec{v} \succ \vec{w})$~~
~~W := W $- \{\vec{w}\}$~~
if $(\neg\exists\vec{w} \in$ W. $\vec{w} \succ \vec{v})$ // $\vec{v}$ survived
W := W $\cup \{\vec{v}\}$ *// if there is room*

$\mathcal{O}(n)$
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 $\mathcal{O}$?)

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 $\mathcal{O}$?)

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 $\mathcal{O}$?)

BNL_R & SFS_R:	Compare $\vec{v}$ against window $\vec{w}$’s in a *random* order.
BNL & **SFS**:	Order window $\vec{w}$’s intelligently to reduce #comparisons.

Analyses of #Comparisons

new!

BNL$_R$:

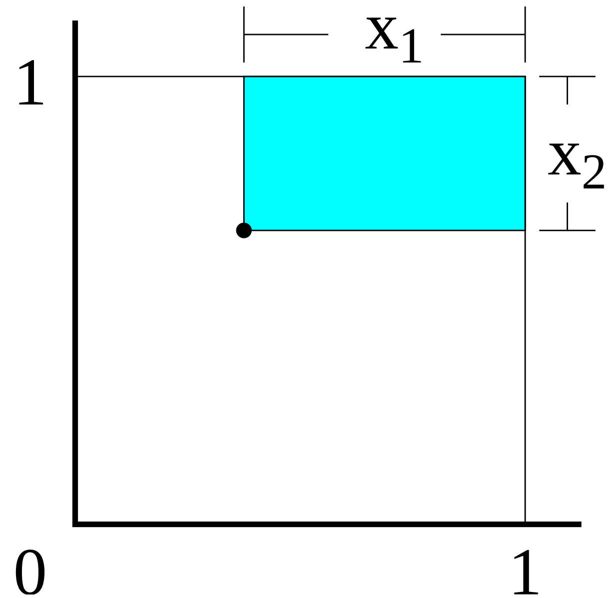
$$\sum_{i=0}^{n-1} \int_{x_k=0}^{1} \int_{x_{k-1}=0}^{1} \dots \int_{x_1=0}^{1} \widehat{\textbf{mttf}}_k(x_1 \cdot \ldots \cdot x_k, i) dx_1 \ldots dx_k$$

Analyses of #Comparisons

new!

BNL$_R$:

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



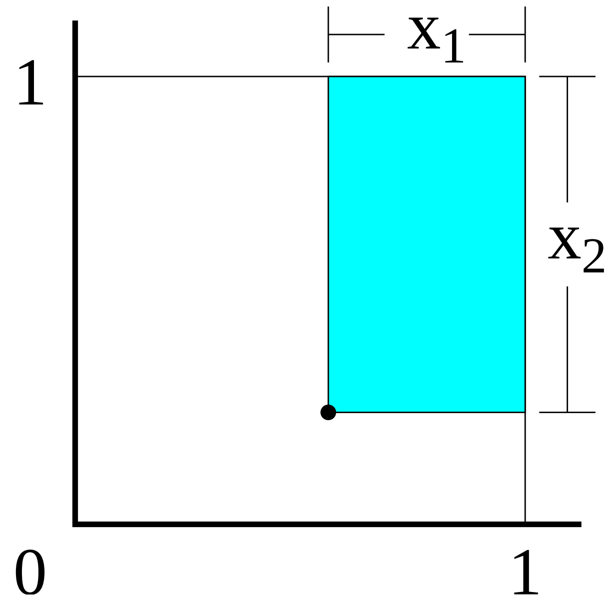
mttf: "mean time to failure"

Analyses of #Comparisons

new!

BNL$_R$:

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



mttf: "mean time to failure"

Analyses of #Comparisons

new!

BNL$_R$:

$$\sum_{i=0}^{n-1} \int_{x_k=0}^{1} \int_{x_{k-1}=0}^{1} \dots \int_{x_1=0}^{1} \widehat{\mathbf{mttf}}_k(x_1 \cdot \ldots \cdot x_k, i) dx_1 \ldots dx_k$$

Analyses of #Comparisons

new!

BNL$_R$:

$$\int_{z=0}^{1}\int_{x_k=0}^{1}\int_{x_{k-1}=0}^{1}\ldots\int_{x_1=0}^{1}\widehat{\textbf{mttf}}_k(x_1\cdot\ldots\cdot x_k, zn)dx_1\ldots dx_k dz$$

Analyses of #Comparisons

new!

BNL$_R$:

$$\int_{z=0}^{1}\int_{x_k=0}^{1}\int_{x_{k-1}=0}^{1}\dots\int_{x_1=0}^{1}\widehat{\mathbf{mttf}}_k(x_1\cdot\ldots\cdot x_k, zn)dx_1\ldots dx_k dz$$

SFS$_R$ w/o elimination from window:

$$\int_{z=0}^{1}\int_{x_{k-1}=0}^{1}\dots\int_{x_1=0}^{1}\widehat{\mathbf{mttf}}_k(x_1\cdot\ldots\cdot x_{k-1}, zn)dx_1\ldots dx_{k-1} dz$$

Analyses of #Comparisons

new!

BNL$_R$:

$$\int_{z=0}^{1}\int_{x_k=0}^{1}\int_{x_{k-1}=0}^{1}\cdots\int_{x_1=0}^{1}\widehat{\mathbf{mttf}}_k(x_1\cdot\ldots\cdot x_k, zn)dx_1\ldots dx_k dz$$

SFS$_R$ w/o elimination from window:

$$\int_{z=0}^{1}\int_{x_{k-1}=0}^{1}\cdots\int_{x_1=0}^{1}\widehat{\mathbf{mttf}}_k(x_1\cdot\ldots\cdot x_{k-1}, zn)dx_1\ldots dx_{k-1} dz$$

SFS$_R$ w/ elimination from window:

$$\int_{z=0}^{1}\int_{x_{k-1}=0}^{1}\cdots\int_{x_1=0}^{1}\widehat{\mathbf{mttf}}_{k-1}(x_1\cdot\ldots\cdot x_{k-1}, zn)dx_1\ldots dx_{k-1} dz$$

Analyses of #Comparisons

new!

BNL$_R$:

$$\int_{z=0}^{1}\int_{x_k=0}^{1}\int_{x_{k-1}=0}^{1}\dots\int_{x_1=0}^{1}\widehat{\mathbf{mttf}}_k(x_1\cdot\ldots\cdot x_k, zn)dx_1\ldots dx_k dz$$

SFS$_R$ w/o elimination from window:

$$\int_{z=0}^{1}\int_{x_{k-1}=0}^{1}\dots\int_{x_1=0}^{1}\widehat{\mathbf{mttf}}_k(x_1\cdot\ldots\cdot x_{k-1}, zn)dx_1\ldots dx_{k-1} dz$$

SFS$_R$ w/ elimination from window:

$$\int_{z=0}^{1}\int_{x_{k-1}=0}^{1}\dots\int_{x_1=0}^{1}\widehat{\mathbf{mttf}}_{k-1}(x_1\cdot\ldots\cdot x_{k-1}, zn)dx_1\ldots dx_{k-1} dz$$

SFS effectively saves “one dimension” over BNL.

Analyses of #Comparisons

new!

Results

$$\widehat{\mathsf{mttf}}_k(x, n) \approx \frac{\mathsf{H}_{k-1,n}}{\mathsf{H}_{k-1,xn}}$$

These converge in the limit.

Analyses of #Comparisons

new!

Results

$$\widehat{\textbf{mttf}}_k(x, n) \approx \frac{\mathsf{H}_{k-1,n}}{\mathsf{H}_{k-1,xn}}$$

These converge in the limit.

Analytical solution matches observation.

Analyses of #Comparisons

new!

Results

$$\widehat{\textbf{mttf}}_k(x, n) \approx \frac{\mathsf{H}_{k-1,n}}{\mathsf{H}_{k-1,xn}}$$

These converge in the limit.

Analytical solution matches observation.

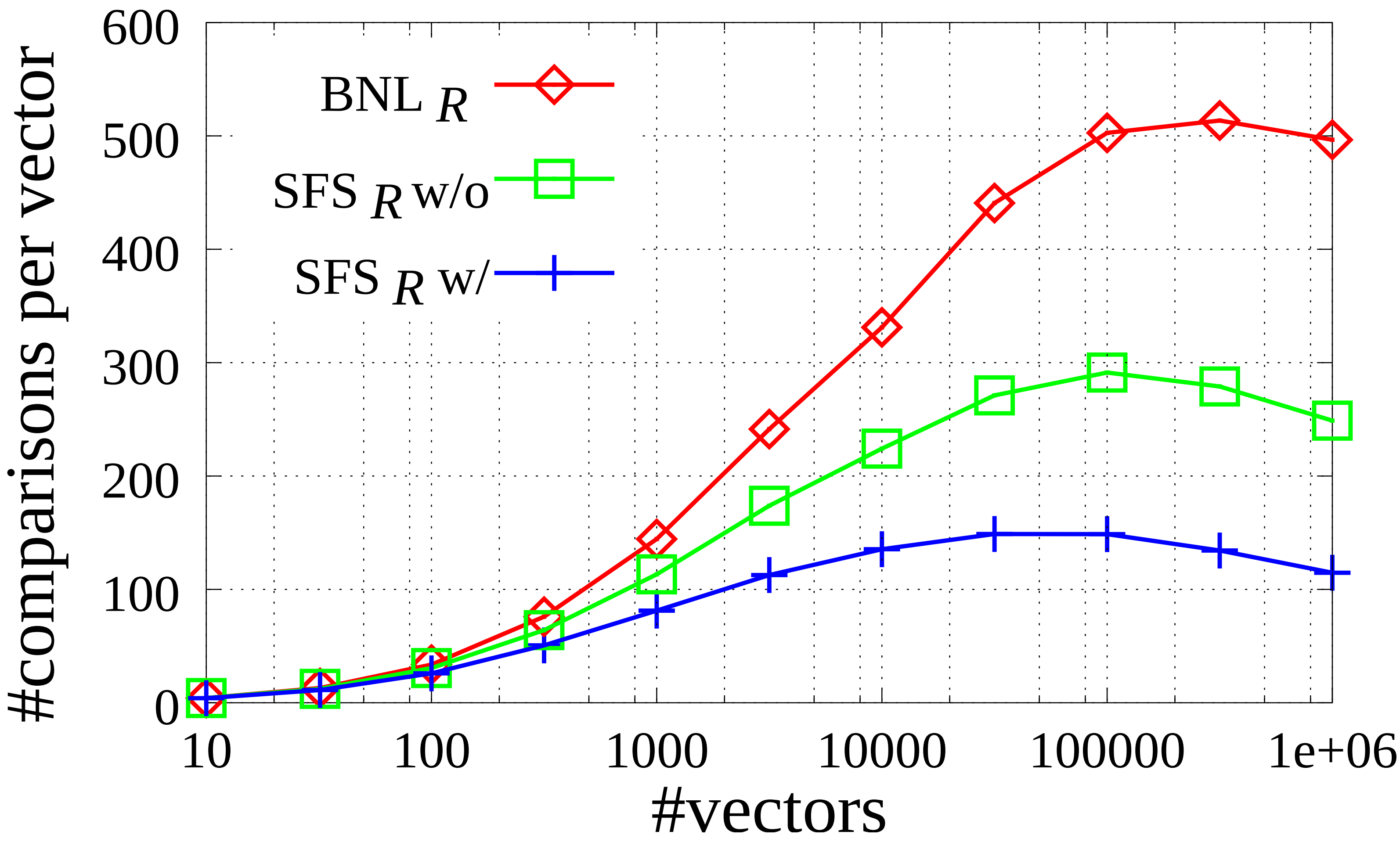
Thm. Under CI, BNL_R and SFS_R are $\mathcal{O}(n)$ average case.

Proof.

$$\lim_{n \to \infty} \int_{z=0}^{1} \int_{x_k=0}^{1} \ldots \int_{x_1=0}^{1} \widehat{\textbf{mttf}}_k(\ldots, zn) d \ldots = 1$$

BNL & SFS

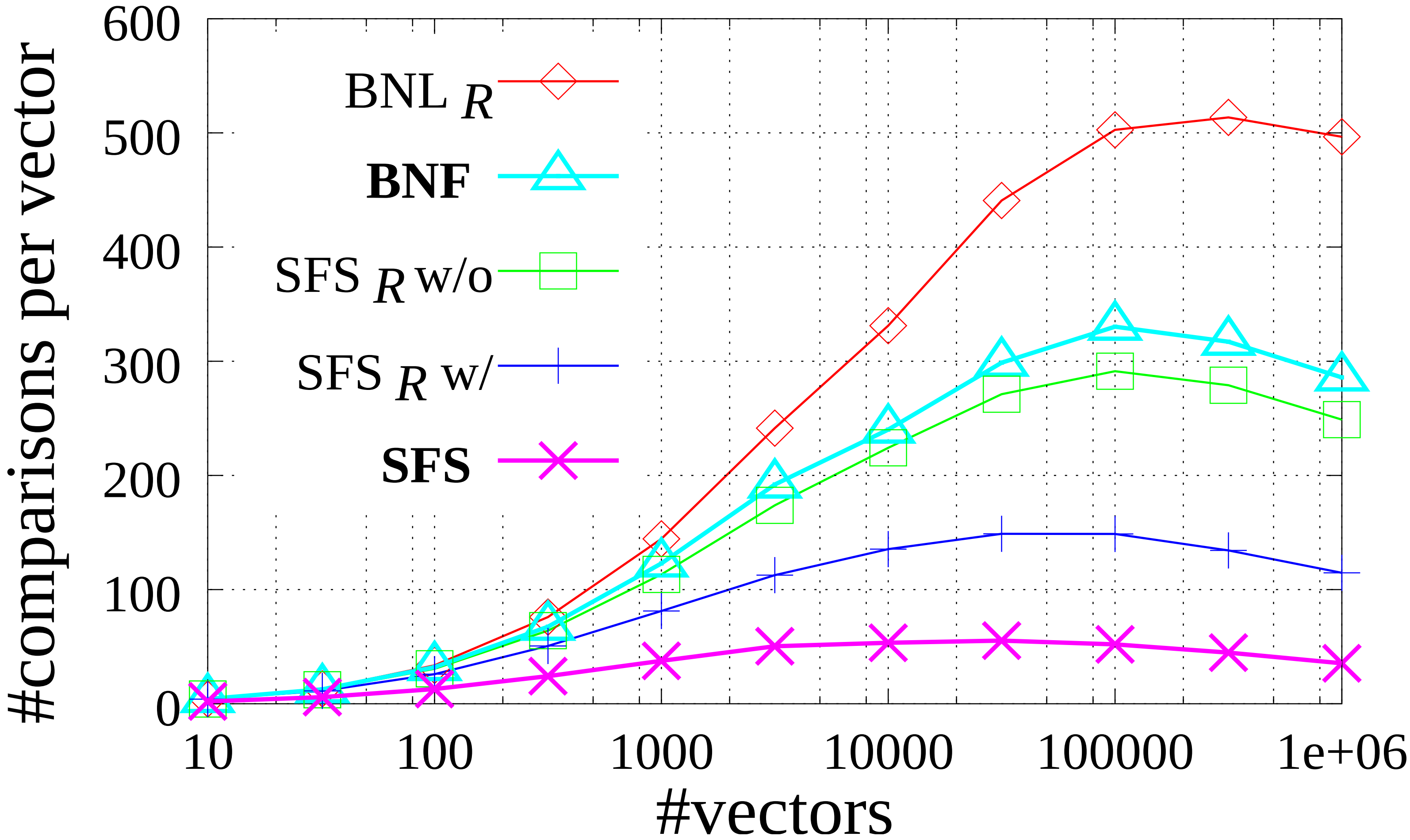
Comparisons per Vector



$k = 7$

BNL & SFS

Comparisons per Vector



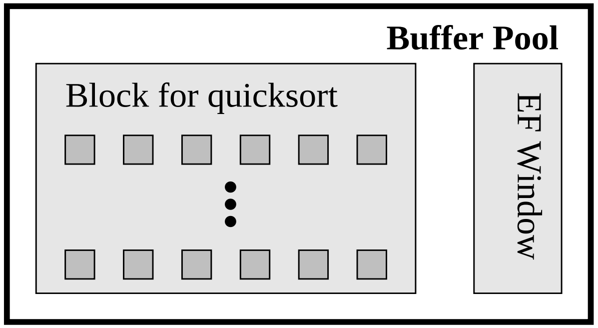
$$k = 7$$

IV. The LESS Algorithm

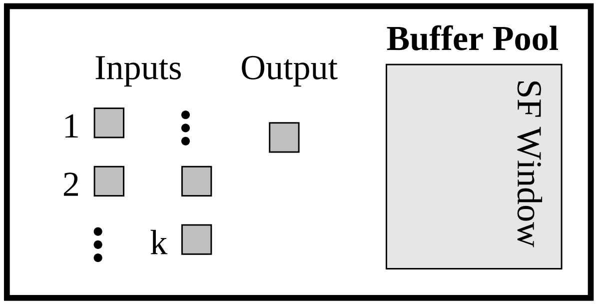
Description

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)
  - ⋮

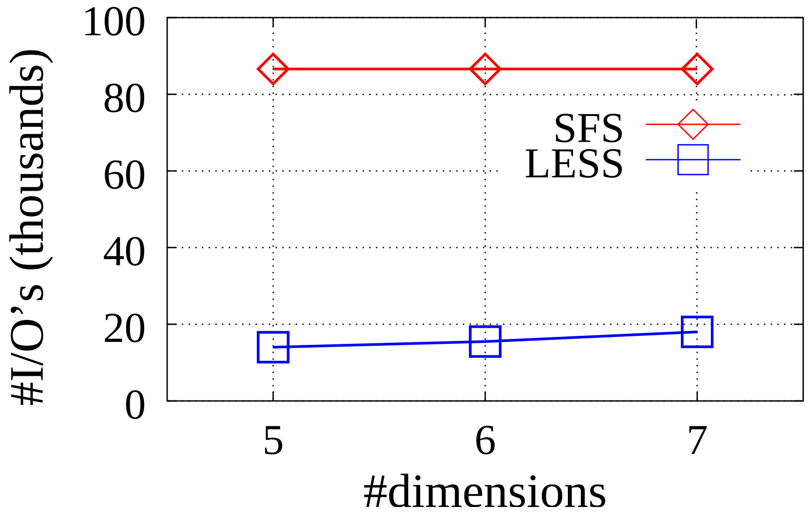


block-sort pass

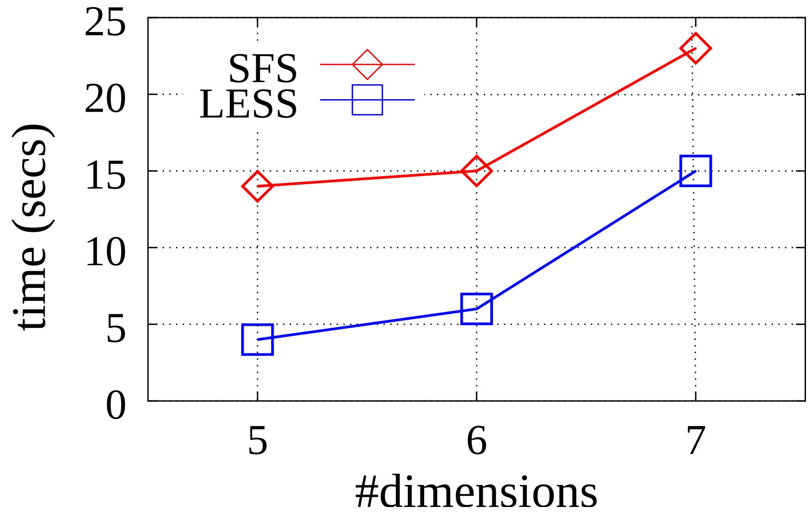


last merge pass

LESS: Performance



I/O's



time

$n = 500,000$
EF window: 200 vectors
SF window: 76 pages, ~3,000 vectors
Pentium III, 733 MHz
RedHat Linux 7.3

LESS: Linear Average-Case

Summary

$\mathcal{O}(n)$ **average-case run-time** (under UI, **Thm. 13**)

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

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.

V. 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?

In Closing. . .

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

```
select C_1, ..., C_j,        – columns to keep
       D_1, ..., D_k,        – skyline dimensions (MAX assumed)
       E_1, ..., E_l         – DIFF columns
    from OurTable
    except
    select X.C_1, ..., X.C_j,
           X.D_1, ..., X.D_k,
           X.E_1, ..., X.E_l
        from OurTable X, OurTable Y
        where Y.D_1 ≥ X.D_1 and ... Y.D_k ≥ X.D_k and
              (Y.D_1 > X.D_1 or ... Y.D_k > X.D_k) and
              Y.E_1 = X.E_1 and ... Y.E_l = X.E_l
```

Certainly $\mathcal{O}(n^2)$, even for average-case.

Skyline Cardinality

harmonic numbers [Godfrey 2004 (FoIKS)]

1. The *harmonic of* n, for $n > 0$: $\mathsf{H}_n = \sum_{i=1}^{n} \frac{1}{i}$
2. The k*-th order harmonic of* n, for integers $k > 0$ and integers $n > 0$: $\mathsf{H}_{k,n} = \sum_{i=1}^{n} \frac{\mathsf{H}_{k-1,i}}{i}$

   Define $\mathsf{H}_{0,n} = 1$, for $n > 0$. Define $\mathsf{H}_{k,0} = 0$, for $k > 0$.
3. The k*-th hyper-harmonic of* n, for integers $k > 0$ and integers $n > 0$: $\mathcal{H}_{k,n} = \sum_{i=1}^{n} \frac{1}{i^k}$

$$\widehat{m}_{k+1,n} = \mathsf{H}_{k,n} = \sum_{i_1=1}^{n} \sum_{i_2=1}^{i_1} \cdots \sum_{i_k=1}^{i_{k-1}} \frac{1}{i_1 i_2 \cdots i_k}$$

Skyline Cardinality

asymptotic [Godfrey 2004 (FoIKS)]

Thm.

$$\mathsf{H}_{k,n} = \sum_{\substack{c_1,\ldots,c_k \geq 0 \;\wedge \\ 1\cdot c_1 + 2\cdot c_2 + \ldots + k\cdot c_k = k}} \prod_{i=1}^{k} \frac{\mathcal{H}_{i,n}^{c_i}}{i^{c_i} \cdot c_i!}$$

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

Follows from Knuth's generalization via generating functions.

- Only $\mathcal{H}_{1,n}$ $(= \mathsf{H}_n)$ diverges with n.
- Each $\mathcal{H}_{i,n}$ for $i > 1$ converges.
- **Thm.** $\mathsf{H}_{k,n}$ is $\Theta((\ln n)^k / k!)$.
- **Thm.** $\widehat{m}_{k,n}$ is $\Theta((\ln n)^{k-1} / (k-1)!)$.

Skyline Cardinality

examples [Godfrey 2004 (FoIKS)]

- $\mathsf{H}_{2,n} = \frac{1}{2}\mathsf{H}_n^2 + \frac{1}{2}\mathcal{H}_{2,n}$,
- $\mathsf{H}_{3,n} = \frac{1}{6}\mathsf{H}_n^3 + \frac{1}{2}\mathsf{H}_n\mathcal{H}_{2,n} + \frac{1}{3}\mathcal{H}_{3,n}$, and
- $\mathsf{H}_{4,n} = \frac{1}{24}\mathsf{H}_n^4 + \frac{1}{3}\mathsf{H}_n\mathcal{H}_{3,n} + \frac{1}{8}\mathcal{H}_{2,n}^2 + \frac{1}{4}\mathsf{H}_n^2\mathcal{H}_{2,n} + \frac{1}{4}\mathcal{H}_{4,n}$.
- $\ldots$

D&C | +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: $\mathcal{O}(n \lg^{k-2} n)$

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

LD&C

(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!

D&C | —Sort

LD&C

(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?