

SPARK: Top-k Keyword Query in Relational Databases

Wei Wang

weiw at cse.unsw.edu.au

<http://www.cse.unsw.edu.au/~weiw>

School of Computer Science & Engineering
University of New South Wales

16 July, 2007



UNSW

Outline

- 1 Introduction
- 2 Ranking Search Results
- 3 Efficient Query Processing
- 4 Experiments
- 5 Related Work
- 6 Conclusions

- My research revolves around *query processing & optimization* issues.
- Research interests:
 - DB + IR
 - Data Warehousing, Data Integration, and Data Mining
 - XML Query Processing & Optimization
 - Spatial Databases
- This talk is based on our recent SIGMOD07 paper.

- 1 Introduction
- 2 Ranking Search Results
- 3 Efficient Query Processing
- 4 Experiments
- 5 Related Work
- 6 Conclusions

Search Without Boundaries

- Example:
 - Fail to locate “World Wide Web” in the following paragraph.

```
... paradigm and have made the vast
amount of information on the World Wide
Web available to even casual users.
...
```

- Should search within the scope of a “logical unit”.
- Non-trivial to support such search
 - Use regular expressions? ⇐ only works for linear layout of texts
 - Find individual matches and grow into “clusters”
 - Scale to disk-resident data?
- Other issues that matter
 - Ranking
 - Efficiency

Keyword Search in Relational Databases

- What is the “logical unit” in databases?

Query: `netvista maxtor`

Keyword Search in Relational Databases

- What is the “logical unit” in databases?

Query: **netvista** **maxtor**

COMPLAINT				
	prodID	custID	comment	solution
c ₁	P121	C111	“disk crashed after ... on an IBM Netvista X41”	“Change Maxtor HD to Seagate HD”
c ₂	P131	C222	“lower-end IBM Netvista caught fire, ...”	“dial 911”
c ₃	P131	C333	“IBM Netvista unstable with Maxtor HD”	“Really?”

PRODUCT			
	prodID	model	description
p ₁	P121	“D540X”	“1.7G CPU ..., Maxtor 80G HD”
p ₂	P131	“D710P”	“1.5G CPU ..., Maxtor 200G HD”

- **Top-5 results:** c₃, c₁, c₃ → p₂, c₁ → p₁, c₂ → p₂
- Results in the form of **JTT** (Joined Tuple Tree)

- Keyword search in DB is good for average users:
 - *“What’s SQL?”*
 - *“I don’t know what’s the cryptic attribute names in your database, but can’t you just show me all you know about John?”*
 - *“I am looking for a movie, but I cannot remember its name, etc. All I know is that one actor is named Jim and the film was probably out in 1995”*
- Other applications:
 - Enterprise search
 - CMS
 - CRM
 - Explorative / interactive data analysis

Search the IMDb

All

Other Searches

- 2001 hanks
- Characters
- Plots
- Biographies
- Guides

[more...](#)

Note: some searches may not yield results

IMDb Search

A search for "2001 hanks" found the following results:
Titles (Approx. Matches) (Displaying 18 Results)

1. **2001: HAL's Legacy** (2001) (TV)
2. **Gigantic Skate Park Tour: Summer 2002** (2002) (TV)
aka *"Tony Hawk's Gigantic Skate Park Tour: Summer 2002"* - USA (complete title)
3. **TV Hanks and Babes 2006** (2006) (TV)
4. **Danish Music Awards 2001** (2001) (TV)
5. **Norah Jones & the Handsome Band Live in 2004** (2004) (V)
6. **Danish music awards 2001 - Fersel med Casper og Lasse** (2001) (TV)
7. **Hand und die 200.000 Kücken** (1952)
8. **Niels Hausgaard live i Cirkusbygningen** (2001) (TV)
aka *"Niels Hausgaard show 2001"* - Denmark (promotional title)
9. **Gotlicher Job, Ein** (2001)
aka *"Jonathan 2001"* - Germany (working title)
10. **Brasilianhas 2001** (2001)
11. **Eurovision 2001: More Than Just a Song Contest** (2001) (TV)
12. **Spølemagisteren 2001** (2001) (TV)
13. **2001 Maniacs** (2005)
14. **Jonas qui aura 25 ans en l'an 2000** (1976)
15. **Hans Warsz - Mein 20. Jahrhundert** (1999)
aka *"Hans Warsz: My 20th Century"* - (English title)
16. **Nárvat Jana z panického exilu do Prahy v léte 2003** (2003)
aka *"Jan's Return from the Parisian Exile to Prague in the Summer of 2003"* - (Engl.)
17. **Starz 2001 - Die Aids-Gala** (2001) (TV)
18. **Cincom titi SNK 2. Mirisosa falansa 2001_nen** (2001) (VG)
aka *"Capcom vs SNK 2: Mark of the Millennium 2001"* - USA

Companies (Approx. Matches) (Displaying 2 Results)

1. **Hans Christian Andersen 2005 Fonden** [dk] (Miscellaneous)
2. **Rhapsody 2001 Inc.** (Production)

Query Results - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

Mondial DBLP IMDb Northwind

2001 Hanks [Advanced Search](#)

IMDb Results 1 - 10 in 40.54 for "2001 Hanks"

	movies : NAME: "Primetime Glick" (2001) {Tom Hanks/Ben Stiller (#2.1)}	31.54 <input type="button" value="Q"/>
	actors : NAME: Hanks , Tom actorplay : CHARACTER: Himself	19.77 <input type="button" value="Q"/>
	movies : NAME: "Primetime Glick" (2001) {Tom Hanks/Ben Stiller (#2.1)}	
	actors : NAME: Hanks , Colin actorplay : CHARACTER: Alex Whitman (1999-2001)	17.87 <input type="button" value="Q"/>
	actorplay : CHARACTER: John Hanks movies : NAME: Rosamunde Pilcher - Wind über dem Fluss (2001) (TV)	17.87 <input type="button" value="Q"/>
	actorplay : CHARACTER: Hanks (Lt./Cmdr./Capt. Hanks)	15.69 <input type="button" value="Q"/>

Done

Search the IMDb

All go

[More searches](#) | [Tips](#)
[IMDbPro.com free trial](#)

Other Searches

2001 hanks

- [Characters](#)
- [Plots](#)
- [Biographies](#)
- [Quotes](#)

[more »](#)

Note: some searches may not yield results

IMDb Search

A search for "2001 hanks" found the following results:


Titles (Approx Matches) (Displaying 18 Results)

- [2001: HAL's Legacy](#) (2001) (TV)
- [Gigantic Skate Park Tour: Summer 2002](#) (2002) (TV)
aka *"Tony Hawk's Gigantic Skate Park Tour: Summer 2002"* - USA (complete title)
- [TV Hunks and Babes 2006](#) (2006) (TV)
- [Danish Music Awards 2001](#) (2001) (TV)
- [Norah Jones & the Handsome Band: Live in 2004](#) (2004) (V)
- [Danish music awards 2001 - Forspil med Casper og Lasse](#) (2001) (TV)
- [Hansl und die 200.000 Kücken](#) (1952)
- [Niels Hausgaard live i Cirkusbygningen](#) (2001) (TV)
aka *"Niels Hausgaard show 2001"* - Denmark (promotional title)
- [Göttlicher Job, Ein](#) (2001)
aka *"Jonathan 2001"* - Germany (working title)
- [Brasileirinhas 2001](#) (2001)
- [Eurovision 2001: More Than Just a Song Contest](#) (2001) (TV)
- [Spellemansprisen 2001](#) (2001) (TV)
- [2001 Maniacs](#) (2005)
- [Jonas qui aura 25 ans en l'an 2000](#) (1976)
- [Hans Warns - Mein 20. Jahrhundert](#) (1999)
aka *"Hans Warns: My 20th Century"* - (English title)
- [Návrat Jana z parížského exilu do Prahy v léte 2003](#) (2003)
aka *"Jan's Return from the Parisian Exile to Prague in the Summer of 2003"* - (English title)
- [Stars 2001 - Die Aids-Gala](#) (2001) (TV)
- [Capcom tai SNK 2: Mirionea faitingu 2001-nen](#) (2001) (VG)
aka *"Capcom vs SNK 2: Mark of the Millennium 2001"* - USA

Companies (Approx Matches) (Displaying 2 Results)

- [Hans Christian Andersen 2005 Fonden](#) [dk] (Miscellaneous)
- [Rhapsody 2001 Inc.](#) (Production)

File Edit View Go Bookmarks

 **SPARK**
searching, pro

IMDb

movies : NAME: "

actors : NAME: H
actorplay : CH
movies : N
(#2.1)}

actors : NAME: H
actorplay : CH

actorplay : CHAF
movies : NAM

actorplay : CHAF

Done

Query Results - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

 **SPARK**
searching, probing & ranking

Mondial [DBLP](#) [IMDb](#) [Northwind](#)

2001 Hanks

[Advanced Search](#)

IMDB Results 1 - 10 in " 40.54 for " 2001 Hanks "

	movies : NAME: "Primetime Glick" (2001) { Tom Hanks /Ben Stiller (#2.1)}	31.54 
	actors : NAME: Hanks , Tom	19.77 
	actorplay : CHARACTER: Himself	
	movies : NAME: "Primetime Glick" (2001) { Tom Hanks /Ben Stiller (#2.1)}	
	actors : NAME: Hanks , Colin	17.87 
	actorplay : CHARACTER: Alex Whitman (1999- 2001)	
	actorplay : CHARACTER: John Hanks	17.87 
	movies : NAME: Rosamunde Pilcher - Wind über dem Fluss (2001) (TV)	
	actorplay : CHARACTER: Hanks (Lt./Cmdr./Capt. Hanks)	15.69 

Done

Top-k Keyword Search

- Input:
 - Relational schema:
 $\mathcal{R} = \{R_1, R_2, \dots, R_{|\mathcal{R}|}\}$
 - Query keywords:
 $Q = \{q_1, q_2, \dots, q_{|Q|}\}$
 - a constant: k
 - a scoring function: $sc(T, Q)$
- Output:
 - k **JTT**s, T_1, T_2, \dots, T_k , with the highest scores.

Challenges:

Running Example

Input:

- $\mathcal{R} =$
 $\{\text{COMPLAINT}, \text{PRODUCT}\}$
- $Q = \{\text{netvista}, \text{maxtor}\}$
- $k = 3$

Output:

- $c_3(4.3), c_1(4.0), c_3 \rightarrow p_2(2.9)$

Top-k Keyword Search

- Input:

- Relational schema:

$$\mathcal{R} = \{ R_1, R_2, \dots, R_{|\mathcal{R}|} \}$$

- Query keywords:

$$Q = \{ q_1, q_2, \dots, q_{|Q|} \}$$

- a constant: k

- a **scoring function**: $sc(T, Q)$

- Output:

- k **JTTs**, T_1, T_2, \dots, T_k , with the highest scores.

Running Example

Input:

- $\mathcal{R} =$
 $\{ \text{COMPLAINT}, \text{PRODUCT} \}$
- $Q = \{ \text{netvista}, \text{maxtor} \}$
- $k = 3$

Output:

- $c_3(4.3), c_1(4.0), c_3 \rightarrow p_2(2.9)$

Challenges:

- How to ensure the ranked output agrees with human judgement?
- How to retrieve such results efficiently?

- 1 Introduction
- 2 Ranking Search Results**
- 3 Efficient Query Processing
- 4 Experiments
- 5 Related Work
- 6 Conclusions

Our Proposed Ranking Function

$$\begin{aligned} \text{sc}(T, Q) &= \text{sc}_a(T, Q) && // \text{ IR score} \\ &\times \text{sc}_b(T, Q) && // \text{ Completeness} \\ &\times \text{sc}_c(T, Q) && // \text{ Size normalization} \end{aligned}$$

Score Function Part A — IR-style Ranking

Example: Which one is better, $c_3 \rightarrow p_2$ or $c_2 \rightarrow p_2$?

COMPLAINT				
	prodID	custID	comment	solution
c_2	P131	C222	"lower-end IBM Netvista caught fire, ..."	"dial 911"
c_3	P131	C333	"IBM Netvista unstable with Maxtor HD"	"Really?"

PRODUCT			
	prodID	model	description
p_2	P131	"D710P	"1.5G CPU ... , Maxtor 200G HD"

Score Function Part A — IR-style Ranking

Example: Which one is better, $c_3 \rightarrow p_2$ or $c_2 \rightarrow p_2$?

COMPLAINT				
	prodID	custID	comment	solution
c_2	P131	C222	“lower-end IBM Netvista caught fire, ...”	“dial 911”
c_3	P131	C333	“IBM Netvista unstable with Maxtor HD”	“Really?”

PRODUCT			
	prodID	model	description
p_2	P131	“D710P”	“1.5G CPU ..., Maxtor 200G HD”

- 1 Model JTT as a *virtual document*
- 2 Apply IR score functions

$$sc_a(T, Q) = \sum_{t \in Q \cap D} \left(\frac{1 + \ln(1 + \ln(tf))}{1 - s + s \cdot \frac{df}{\text{avdl}}} \cdot \ln \frac{N+1}{df} \right)$$

	tf _{netvista}	tf _{maxtor}	$sc_a(T, Q)$
$c_3 \rightarrow p_2$	1	2	1.71
$c_2 \rightarrow p_2$	1	1	1.26

Score Function Part B — Completeness Factor

Example: Which one is better, $c_2 \rightarrow p_2$ or $c'_3 \rightarrow p_2$?

COMPLAINT				
	prodID	custID	comment	solution
c_2	P131	C222	"lower-end IBM Netvista caught fire, ..."	"dial 911"
c'_3	P131	C333	"IBM ThinkPad X41 unstable with Maxtor HD"	"Really?"

PRODUCT			
	prodID	model	description
p_2	P131	"D710P"	"1.5G CPU ..., Maxtor 200G HD"

Score Function Part B — Completeness Factor

Example: Which one is better, $c_2 \rightarrow p_2$ or $c'_3 \rightarrow p_2$?

COMPLAINT				
	prodID	custID	comment	solution
c_2	P131	C222	"lower-end IBM Netvista caught fire, ..."	"dial 911"
c'_3	P131	C333	"IBM ThinkPad X41 unstable with Maxtor HD"	"Really?"

PRODUCT			
	prodID	model	description
p_2	P131	"D710P"	"1.5G CPU ..., Maxtor 200G HD"

- 1 Intuition: for short queries, user prefer results matching *more* (distinct) keywords.
- 2 Derived from extended Boolean model:

$$sc_b(T, Q) = 1 - \left(\frac{\sum_{1 \leq i \leq m} (1 - T_i)^p}{m} \right)^{\frac{1}{p}}$$

	tf _{netvista}	tf _{maxtor}	sc _b (T, Q)
$c_2 \rightarrow p_2$	1	1	0.65
$c'_3 \rightarrow p_2$	0	2	0.29

Score Function Part C — Size Normalization

Example: Which one is better, c_3 or $c_2 \rightarrow p_2$?

COMPLAINT				
	prodID	custID	comment	solution
c_2	P131	C222	"lower-end IBM Netvista caught fire, ..."	"dial 911"
c_3	P131	C333	"IBM Netvista unstable with Maxtor HD"	"Really?"

PRODUCT			
	prodID	model	description
p_2	P131	"D710P"	"1.5G CPU ... , Maxtor 200G HD"

Score Function Part C — Size Normalization

Example: Which one is better, c_3 or $c_2 \rightarrow p_2$?

COMPLAINT				
	prodID	custID	comment	solution
c_2	P131	C222	“lower-end IBM Netvista caught fire, ...”	“dial 911”
c_3	P131	C333	“IBM Netvista unstable with Maxtor HD”	“Really?”

PRODUCT			
	prodID	model	description
p_2	P131	“D710P”	“1.5G CPU ..., Maxtor 200G HD”

- 1 Size of a JTT matters!
- 2 $sc_c(T, Q) = (1 + s_1 - s_1 \cdot \text{size}(\text{CN})) \cdot (1 + s_2 - s_2 \cdot \text{size}(\text{CN}^{\text{nf}}))$

	$sc_a(T, Q)$	$sc_b(T, Q)$	$sc_c(T, Q)$
c_3	1.26	0.75	1.00
$c_2 \rightarrow p_2$	1.26	0.75	0.56

Effectiveness

- Test the effectiveness against previous methods.

Table: Top-1 Result for Query “nikos clique” on DBLP

Method	Top-1 Result
[Hristidis VLDB03]	InProceeding: <u>Clique-to-Clique</u> Distance ...
[Liu SIGMOD06]	<pre>graph TD; Series --> P1[Proceeding]; Series --> P2[Proceeding]; P1 --> R1[Person: Nikos Karatzas]; P2 --> R2[InProceeding: Maximum Clique Transversals]; P2 --> R3[InProceeding: On ... Clique-Width and ...];</pre>
Ours	Person: <u>Nikos Mamoulis</u> ← RPI → InProceeding: Constraint-Based Algorithms for Computing <u>Clique</u> Intersection Joins

- 1 Introduction
- 2 Ranking Search Results
- 3 Efficient Query Processing**
- 4 Experiments
- 5 Related Work
- 6 Conclusions

Query Processing

Three steps:

- 1 Generate candidate tuples in every relation in the schema (using full-text indexes)

Running Example

$$C^Q = [c_3, c_2]$$

$$P^Q = [p_3]$$

COMPLAINT				
	prodID	custID	comment	solution
c ₂	P131	C222	"lower-end IBM Netvista caught fire, ..."	"dial 911"
c ₃	P131	C333	"IBM Netvista unstable with Maxtor HD"	"Really?"

PRODUCT			
	prodID	model	description
p ₂	P131	"D710P"	"1.5G CPU ..., Maxtor 200G HD"

Three steps:

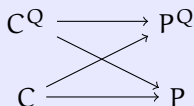
- 2 Enumerate all possible Candidate Networks (CN) using a breath-first subgraph enumeration algorithm.

Running Example

“Valid” CN of size ≤ 3 :

- C^Q
- P^Q
- $C^Q \rightarrow P^Q$
- $C^Q \rightarrow P \leftarrow C^Q$
- $C^Q \rightarrow P^Q \leftarrow C^Q$

Note that $P^Q \leftarrow C^* \rightarrow P^Q$ is *not* a valid CN.



Query Processing ...

Three steps:

- Execute the CNs
 - Most algorithms differ here
 - Naive alg: execute every CN.
 - The key is how to optimize for the top-k retrieval.

Running Example

- C^Q
- P^Q
- $C^Q \rightarrow P^Q$
- $C^Q \rightarrow P \leftarrow C^Q$
- $C^Q \rightarrow P^Q \leftarrow C^Q$

```
SELECT * FROM C, P
WHERE C.prodID = P.prodID
```

What if $k \ll$ (size of join results)

Analogy: Mine Sweeper

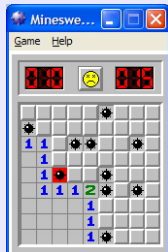
CN $C^Q \rightarrow P^Q$ (easily
generalized to CNs with
> 2 joins)

However, such probes are very
expensive!

Query Processing ...

Three steps:

- Execute the CNs
 - Most algorithms differ here
 - Naive alg: execute every CN.
 - The key is how to optimize for the top-k retrieval.



Analogy: Mine Sweeper

CN $C^Q \rightarrow P^Q$ (easily generalized to CNs with > 2 joins)

Running Example

- C^Q
- P^Q
- $C^Q \rightarrow P^Q$
- $C^Q \rightarrow P \leftarrow C^Q$
- $C^Q \rightarrow P^Q \leftarrow C^Q$

```
SELECT * FROM C, P
WHERE C.prodID = P.prodID
```

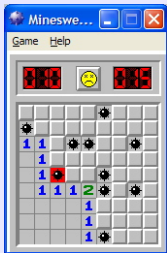
What if $k \ll$ (size of join results)

However, such probes are very **expensive!**

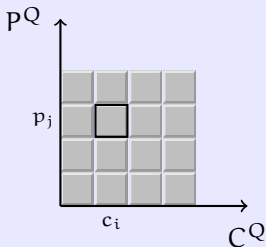
Query Processing ...

Three steps:

- 3 Execute the CNs
 - Most algorithms differ here
 - Naive alg: execute every CN.
 - The key is how to optimize for the top-k retrieval.



Analogy: Mine Sweeper



CN $C^Q \rightarrow P^Q$ (easily generalized to CNs with > 2 joins)

Running Example

- C^Q
- P^Q
- $C^Q \rightarrow P^Q$
- $C^Q \rightarrow P \leftarrow C^Q$
- $C^Q \rightarrow P^Q \leftarrow C^Q$

```
SELECT * FROM C, P
WHERE C.prodID = P.prodID
```

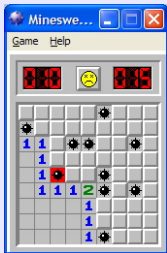
What if $k \ll$ (size of join results)

However, such probes are very **expensive!**

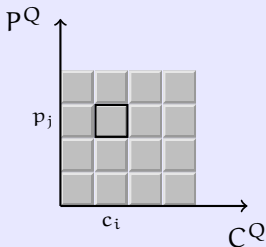
Query Processing ...

Three steps:

- Execute the CNs
 - Most algorithms differ here
 - Naive alg: execute every CN.
 - The key is how to optimize for the top-k retrieval.



Analogy: Mine Sweeper



CN $C^Q \rightarrow P^Q$ (easily generalized to CNs with > 2 joins)

Running Example

- C^Q
- P^Q
- $C^Q \rightarrow P^Q$
- $C^Q \rightarrow P \leftarrow C^Q$
- $C^Q \rightarrow P^Q \leftarrow C^Q$

```
SELECT * FROM C, P
WHERE C.prodID = P.prodID
```

What if $k \ll$ (size of join results)

```
SELECT * FROM C, P
WHERE C.prodID = P.prodID
AND C.cid = c_i.cid
AND P.pid = p_j.pid
```

However, such probes are very **expensive!**

It is a Search Problem



An unknown
space to
search

It is a Search Problem



An unknown
space to
search

Many cells
has 0 score

It is a Search Problem



An unknown
space to
search

Many cells
has 0 score

In general, we cannot stop even if we found one result ($k = 1$)

It is a Search Problem



An unknown
space to
search



Many cells
has 0 score



In general, we cannot stop even if we found one result
($k = 1$)



But, if we know the scores^{*} are monotonic wrt both axes

* : non-zero scores

It is a Search Problem



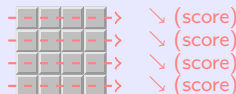
An unknown space to search



Many cells has 0 score



In general, we cannot stop even if we found one result ($k = 1$)



But, if we know the scores* are monotonic wrt both axes

*: non-zero scores

It is a Search Problem



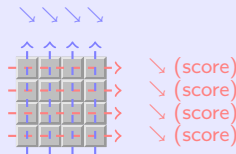
An unknown space to search



Many cells has 0 score



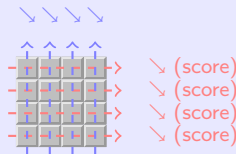
In general, we cannot stop even if we found one result ($k = 1$)



But, if we know the scores* are monotonic wrt both axes

* : non-zero scores

It is a Search Problem



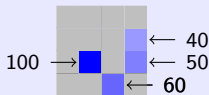
An unknown space to search

Many cells has 0 score

In general, we cannot stop even if we found one result ($k = 1$)

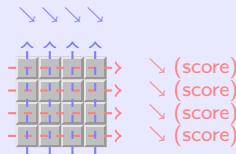
But, if we know the scores* are monotonic wrt both axes

*: non-zero scores



An easier search space to deal with

It is a Search Problem



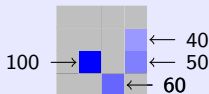
An unknown space to search

Many cells has 0 score

In general, we cannot stop even if we found one result ($k = 1$)

But, if we know the scores* are monotonic wrt both axes

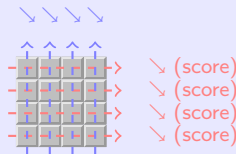
*: non-zero scores



An easier search space to deal with

What's the stopping criterion? (e.g., $k = 1$)?

It is a Search Problem



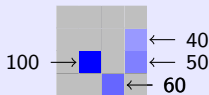
An unknown space to search

Many cells has 0 score

In general, we cannot stop even if we found one result ($k = 1$)

But, if we know the scores* are monotonic wrt both axes

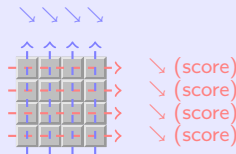
*: non-zero scores



An easier search space to deal with

What's the stopping criterion? (e.g., $k = 1$)?

It is a Search Problem



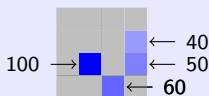
An unknown space to search

Many cells has 0 score

In general, we cannot stop even if we found one result ($k = 1$)

But, if we know the scores* are monotonic wrt both axes

*: non-zero scores



An easier search space to deal with

What's the stopping criterion? (e.g., $k = 1$)?

Lessons learned:

- No optimization exists if we are dealing with a general search space.
- We should stop earlier with the help of a **monotonic** score upper bounding function.
- We should check candidates in an optimal order.

Finding a Monotonic Upper Bounding Function

- It is non-trivial to find a (good) monotonic upper bounding function for our scoring function.
 - Previous method (sorting on local scores) does not work
- Our scores are related to the *distribution* of tf's in a JTT
- Deriving $usc(T, Q)$:
 - $sumidf = \sum_w idf_w$
 - $watf(t) = (\frac{1}{sumidf}) \cdot \sum_w (tf_w(t) \cdot idf_w)$
 - $A = sumidf \cdot (1 + \ln(1 + \ln(\sum_t watf(t))))$
 - $B = sumidf \cdot \sum_t watf(t)$

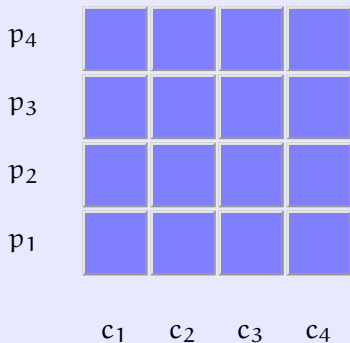
Then,

$$\left. \begin{array}{l} sc_a \leq usc_a = \frac{1}{1-s} \cdot \min(A, B) \\ sc_b = \text{const given the CN} \\ sc_c = \text{const given the CN} \end{array} \right\} \Rightarrow sc(T, Q) \leq usc(T, Q)$$

and $usc(T, Q)$ is monotonic wrt all $watf(t_i)$ ($\forall t_i \in T$)

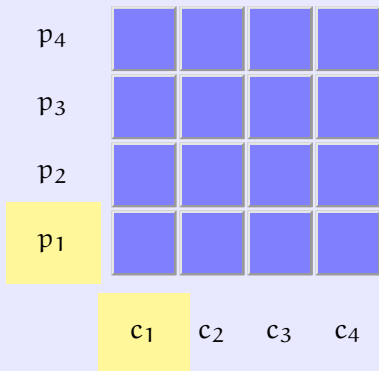
The Single Pipeline Algorithm [Hristidis VLDB03]

- With the discovery of $usc(T, Q)$, the *Single Pipeline* [Hristidis VLDB03] algorithm can be applied.
 - Tuples (on each dimension) sorted in decreasing $watf$ order.
 - Its search strategy is reminiscent of the *Ripple Join*.



The Single Pipeline Algorithm [Hristidis VLDB03]

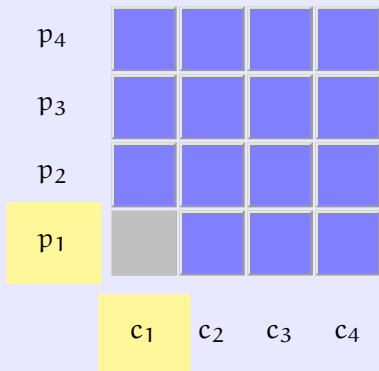
- With the discovery of $usc(T, Q)$, the *Single Pipeline* [Hristidis VLDB03] algorithm can be applied.
 - Tuples (on each dimension) sorted in decreasing $watf$ order.
 - Its search strategy is reminiscent of the *Ripple Join*.



- $c_1 \bowtie p_1$

The Single Pipeline Algorithm [Hristidis VLDB03]

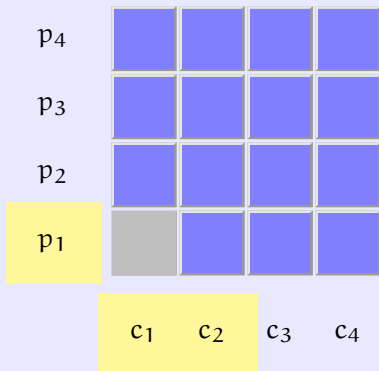
- With the discovery of $usc(T, Q)$, the *Single Pipeline* [Hristidis VLDB03] algorithm can be applied.
 - Tuples (on each dimension) sorted in decreasing $watf$ order.
 - Its search strategy is reminiscent of the *Ripple Join*.



- $c_1 \bowtie p_1$

The Single Pipeline Algorithm [Hristidis VLDB03]

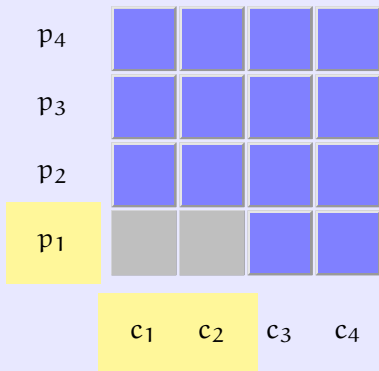
- With the discovery of $usc(T, Q)$, the *Single Pipeline* [Hristidis VLDB03] algorithm can be applied.
 - Tuples (on each dimension) sorted in decreasing $watf$ order.
 - Its search strategy is reminiscent of the *Ripple Join*.



- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_4]$

The Single Pipeline Algorithm [Hristidis VLDB03]

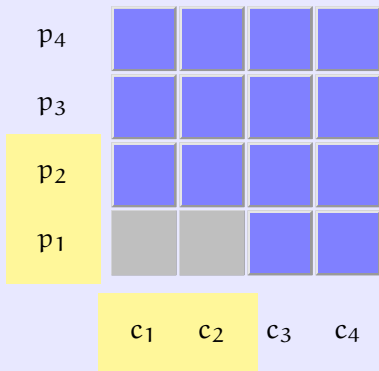
- With the discovery of $usc(T, Q)$, the *Single Pipeline* [Hristidis VLDB03] algorithm can be applied.
 - Tuples (on each dimension) sorted in decreasing $watf$ order.
 - Its search strategy is reminiscent of the *Ripple Join*.



- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_4]$

The Single Pipeline Algorithm [Hristidis VLDB03]

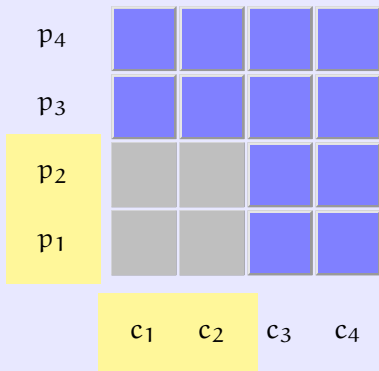
- With the discovery of $usc(T, Q)$, the *Single Pipeline* [Hristidis VLDB03] algorithm can be applied.
 - Tuples (on each dimension) sorted in decreasing $watf$ order.
 - Its search strategy is reminiscent of the *Ripple Join*.



- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_1]$
- $[c_1, \dots, c_2] \bowtie p_2$

The Single Pipeline Algorithm [Hristidis VLDB03]

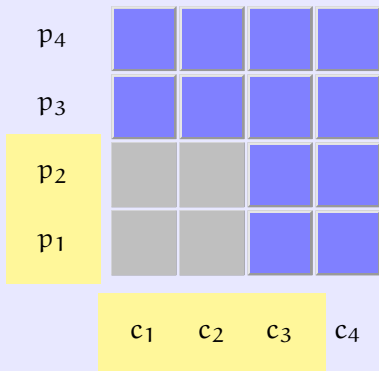
- With the discovery of $usc(T, Q)$, the *Single Pipeline* [Hristidis VLDB03] algorithm can be applied.
 - Tuples (on each dimension) sorted in decreasing $watf$ order.
 - Its search strategy is reminiscent of the *Ripple Join*.



- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_1]$
- $[c_1, \dots, c_2] \bowtie p_2$

The Single Pipeline Algorithm [Hristidis VLDB03]

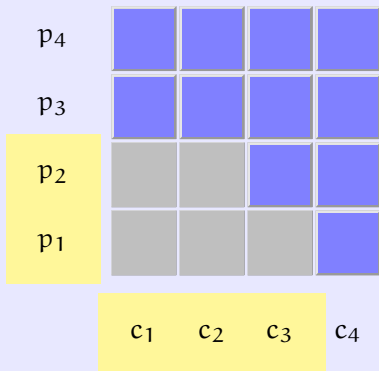
- With the discovery of $usc(T, Q)$, the *Single Pipeline* [Hristidis VLDB03] algorithm can be applied.
 - Tuples (on each dimension) sorted in decreasing $watf$ order.
 - Its search strategy is reminiscent of the *Ripple Join*.



- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_1]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$

The Single Pipeline Algorithm [Hristidis VLDB03]

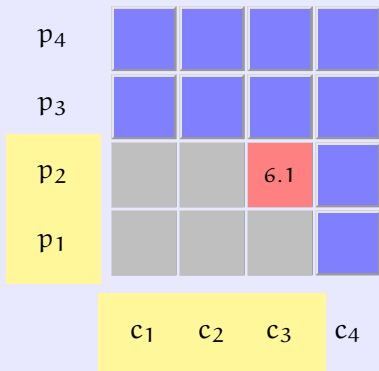
- With the discovery of $usc(T, Q)$, the *Single Pipeline* [Hristidis VLDB03] algorithm can be applied.
 - Tuples (on each dimension) sorted in decreasing $watf$ order.
 - Its search strategy is reminiscent of the *Ripple Join*.



- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_1]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$

The Single Pipeline Algorithm [Hristidis VLDB03]

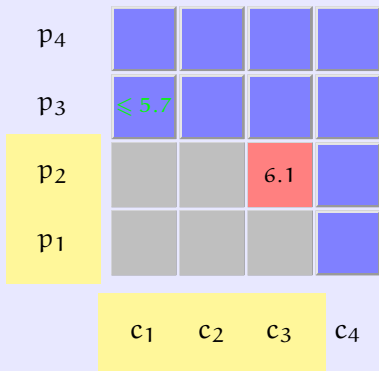
- With the discovery of $usc(T, Q)$, the *Single Pipeline* [Hristidis VLDB03] algorithm can be applied.
 - Tuples (on each dimension) sorted in decreasing $watf$ order.
 - Its search strategy is reminiscent of the *Ripple Join*.



- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_1]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$

The Single Pipeline Algorithm [Hristidis VLDB03]

- With the discovery of $usc(T, Q)$, the *Single Pipeline* [Hristidis VLDB03] algorithm can be applied.
 - Tuples (on each dimension) sorted in decreasing $watf$ order.
 - Its search strategy is reminiscent of the *Ripple Join*.



- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_1]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$

The Single Pipeline Algorithm [Hristidis VLDB03]

- With the discovery of $usc(T, Q)$, the *Single Pipeline* [Hristidis VLDB03] algorithm can be applied.
 - Tuples (on each dimension) sorted in decreasing *watf* order.
 - Its search strategy is reminiscent of the *Ripple Join*.

p ₄	≤ 5.7	≤ 5.7	≤ 5.7	≤ 5.7
p ₃	≤ 5.7	≤ 5.7	≤ 5.7	≤ 5.7
p ₂			6.1	
p ₁				
	c ₁	c ₂	c ₃	c ₄

- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_1]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$

The Single Pipeline Algorithm [Hristidis VLDB03]

- With the discovery of $usc(T, Q)$, the *Single Pipeline* [Hristidis VLDB03] algorithm can be applied.
 - Tuples (on each dimension) sorted in decreasing $watf$ order.
 - Its search strategy is reminiscent of the *Ripple Join*.

p ₄	≤ 5.7	≤ 5.7	≤ 5.7	≤ 5.7
p ₃	≤ 5.7	≤ 5.7	≤ 5.7	≤ 5.7
p ₂			6.1	
p ₁				≤ 4.3
	c ₁	c ₂	c ₃	c ₄

- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_1]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$

The Single Pipeline Algorithm [Hristidis VLDB03]

- With the discovery of $usc(T, Q)$, the *Single Pipeline* [Hristidis VLDB03] algorithm can be applied.
 - Tuples (on each dimension) sorted in decreasing $watf$ order.
 - Its search strategy is reminiscent of the *Ripple Join*.


p ₄	≤ 5.7	≤ 5.7	≤ 5.7	≤ 4.3
p ₃	≤ 5.7	≤ 5.7	≤ 5.7	≤ 4.3
p ₂			6.1	≤ 4.3
p ₁				≤ 4.3
	c ₁	c ₂	c ₃	c ₄

- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_1]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$

The Single Pipeline Algorithm [Hristidis VLDB03]

- With the discovery of $usc(T, Q)$, the *Single Pipeline* [Hristidis VLDB03] algorithm can be applied.
 - Tuples (on each dimension) sorted in decreasing $watf$ order.
 - Its search strategy is reminiscent of the *Ripple Join*.

p ₄	≤ 5.7	≤ 5.7	≤ 5.7	≤ 4.3
p ₃	≤ 5.7	≤ 5.7	≤ 5.7	≤ 4.3
p ₂			6.1	≤ 4.3
p ₁				≤ 4.3
	c ₁	c ₂	c ₃	c ₄

- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_1]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$
- $sc(c_3 \bowtie p_2) \geq MPFS \Rightarrow$ 

Problem of the Single Pipeline Algorithm

- Single Pipeline algorithm is *not* optimal in the number of database probes.
 - Recall that you pay a price for each probe!

p_4				
p_3				
p_2			6.1	
p_1				
	c_1	c_2	c_3	c_4

- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_1]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$

Problem of the Single Pipeline Algorithm

- Single Pipeline algorithm is *not* optimal in the number of database probes.
 - Recall that you pay a price for each probe!

p_4				
p_3				
p_2			6.1	
p_1				≤ 4.3
	c_1	c_2	c_3	c_4

- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_1]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$

Problem of the Single Pipeline Algorithm

- Single Pipeline algorithm is *not* optimal in the number of database probes.
 - Recall that you pay a price for each probe!

p ₄				≤ 4.3
p ₃				≤ 4.3
p ₂			6.1	≤ 4.3
p ₁				≤ 4.3
	c ₁	c ₂	c ₃	c ₄

- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_1]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$

Problem of the Single Pipeline Algorithm

- Single Pipeline algorithm is *not* optimal in the number of database probes.
 - Recall that you pay a price for each probe!

p ₄				≤ 4.3
p ₃	≤ 7.1			≤ 4.3
p ₂			6.1	≤ 4.3
p ₁				≤ 4.3
	c ₁	c ₂	c ₃	c ₄

- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_4]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$

Problem of the Single Pipeline Algorithm

- Single Pipeline algorithm is *not* optimal in the number of database probes.
 - Recall that you pay a price for each probe!

p ₄				≤ 4.3
p ₃	≤ 7.1	≤ 7.1	≤ 7.1	≤ 4.3
p ₂			6.1	≤ 4.3
p ₁				≤ 4.3
	c ₁	c ₂	c ₃	c ₄

- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_3]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$

Problem of the Single Pipeline Algorithm

- Single Pipeline algorithm is *not* optimal in the number of database probes.
 - Recall that you pay a price for each probe!

p ₄				≤ 4.3
p ₃	≤ 7.1	≤ 7.1	≤ 7.1	≤ 4.3
p ₂			6.1	≤ 4.3
p ₁				≤ 4.3
	c ₁	c ₂	c ₃	c ₄

- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_1]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$
- $[c_1, \dots, c_3] \bowtie p_3$

Problem of the Single Pipeline Algorithm

- Single Pipeline algorithm is *not* optimal in the number of database probes.
 - Recall that you pay a price for each probe!

p ₄				≤ 4.3
p ₃	≤ 7.1	≤ 7.1	≤ 7.1	≤ 4.3
p ₂			6.1	≤ 4.3
p ₁				≤ 4.3
	c ₁	c ₂	c ₃	c ₄

- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_1]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$
- $[c_1, \dots, c_3] \bowtie p_3$

Problem of the Single Pipeline Algorithm

- Single Pipeline algorithm is *not* optimal in the number of database probes.
 - Recall that you pay a price for each probe!

p ₄				≤ 4.3
p ₃		≤ 7.1	≤ 7.1	≤ 4.3
p ₂			6.1	≤ 4.3
p ₁				≤ 4.3
	c ₁	c ₂	c ₃	c ₄

- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_3]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$
- $[c_1, \dots, c_3] \bowtie p_3$

Problem of the Single Pipeline Algorithm

- Single Pipeline algorithm is *not* optimal in the number of database probes.
 - Recall that you pay a price for each probe!

p ₄				≤ 4.3
p ₃		4.9	≤ 7.1	≤ 4.3
p ₂			6.1	≤ 4.3
p ₁				≤ 4.3
	c ₁	c ₂	c ₃	c ₄

- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_3]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$
- $[c_1, \dots, c_3] \bowtie p_3$

Problem of the Single Pipeline Algorithm

- Single Pipeline algorithm is *not* optimal in the number of database probes.
 - Recall that you pay a price for each probe!

p ₄				≤ 4.3
p ₃		4.9		≤ 4.3
p ₂			6.1	≤ 4.3
p ₁				≤ 4.3
	c ₁	c ₂	c ₃	c ₄

- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_1]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$
- $[c_1, \dots, c_3] \bowtie p_3$

Problem of the Single Pipeline Algorithm

- Single Pipeline algorithm is *not* optimal in the number of database probes.
 - Recall that you pay a price for each probe!

p ₄	≤ 4.5			≤ 4.3
p ₃		4.9		≤ 4.3
p ₂			6.1	≤ 4.3
p ₁				≤ 4.3
	c ₁	c ₂	c ₃	c ₄

- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_4]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$
- $[c_1, \dots, c_3] \bowtie p_3$

Problem of the Single Pipeline Algorithm

- Single Pipeline algorithm is *not* optimal in the number of database probes.
 - Recall that you pay a price for each probe!

p ₄	≤ 4.5	≤ 4.5	≤ 4.5	≤ 4.3
p ₃		4.9		≤ 4.3
p ₂			6.1	≤ 4.3
p ₁				≤ 4.3
	c ₁	c ₂	c ₃	c ₄

- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_4]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$
- $[c_1, \dots, c_3] \bowtie p_3$

Problem of the Single Pipeline Algorithm

- Single Pipeline algorithm is *not* optimal in the number of database probes.
 - Recall that you pay a price for each probe!

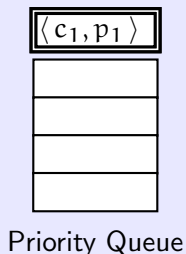
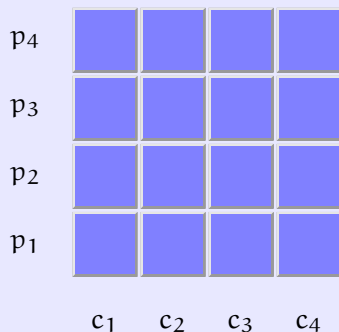
p ₄	≤ 4.5	≤ 4.5	≤ 4.5	≤ 4.3
p ₃		4.9		≤ 4.3
p ₂			6.1	≤ 4.3
p ₁				≤ 4.3
	c ₁	c ₂	c ₃	c ₄

- $c_1 \bowtie p_1$
- $c_2 \bowtie [p_1, \dots, p_1]$
- $[c_1, \dots, c_2] \bowtie p_2$
- $c_3 \bowtie [p_1, \dots, p_2]$
- $[c_1, \dots, c_3] \bowtie p_3$

$c_3 \bowtie p_3$ should **not** have been executed!

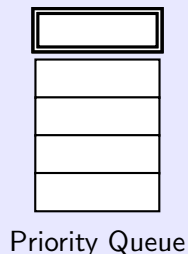
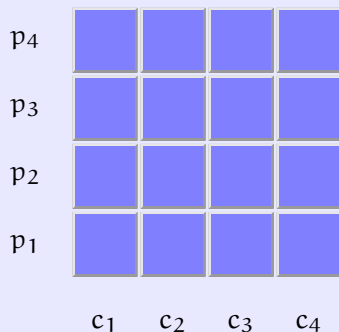
Our Skyline Sweeping Algorithm

- Idea: **be lazy!**
- Based on the **skyline** principle:
 - $usc(c_i \bowtie p_j) \geq usc(c_{i+1} \bowtie p_j)$ and $usc(c_i \bowtie p_j) \geq usc(c_i \bowtie p_{j+1})$
 \Rightarrow if the current “cell” fails, we should *only* examine its *neighbors*, as they are the best possible alternatives.
- Data structure: a priority queue of *candidate cells*
- Its search strategy is reminiscent of the J^* algorithm.



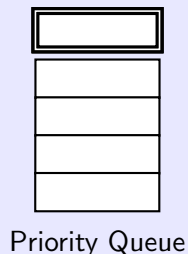
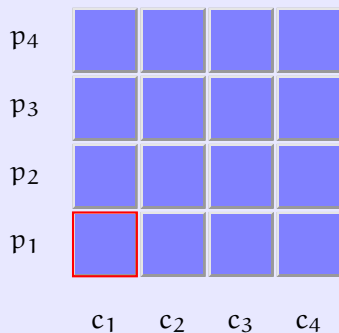
Our Skyline Sweeping Algorithm

- Idea: **be lazy!**
- Based on the **skyline** principle:
 - $usc(c_i \bowtie p_j) \geq usc(c_{i+1} \bowtie p_j)$ and $usc(c_i \bowtie p_j) \geq usc(c_i \bowtie p_{j+1})$
 \Rightarrow if the current “cell” fails, we should *only* examine its *neighbors*, as they are the best possible alternatives.
- Data structure: a priority queue of *candidate cells*
- Its search strategy is reminiscent of the J^* algorithm.



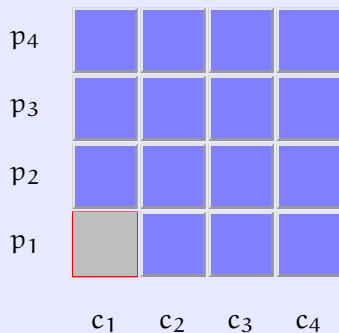
Our Skyline Sweeping Algorithm

- Idea: **be lazy!**
- Based on the **skyline** principle:
 - $usc(c_i \bowtie p_j) \geq usc(c_{i+1} \bowtie p_j)$ and $usc(c_i \bowtie p_j) \geq usc(c_i \bowtie p_{j+1})$
 \Rightarrow if the current “cell” fails, we should *only* examine its *neighbors*, as they are the best possible alternatives.
- Data structure: a priority queue of *candidate cells*
- Its search strategy is reminiscent of the J^* algorithm.

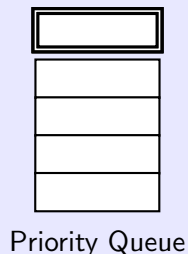


Our Skyline Sweeping Algorithm

- Idea: **be lazy!**
- Based on the **skyline** principle:
 - $usc(c_i \bowtie p_j) \geq usc(c_{i+1} \bowtie p_j)$ and $usc(c_i \bowtie p_j) \geq usc(c_i \bowtie p_{j+1})$
 \Rightarrow if the current “cell” fails, we should *only* examine its *neighbors*, as they are the best possible alternatives.
- Data structure: a priority queue of *candidate cells*
- Its search strategy is reminiscent of the J^* algorithm.

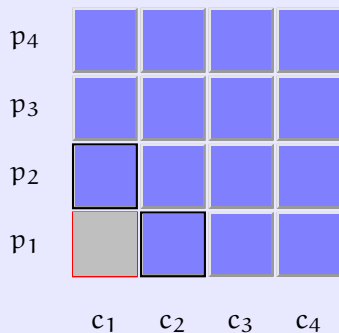


• $c_1 \bowtie p_1$

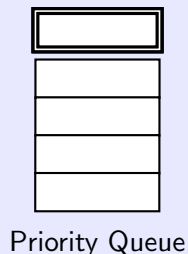


Our Skyline Sweeping Algorithm

- Idea: **be lazy!**
- Based on the **skyline** principle:
 - $usc(c_i \bowtie p_j) \geq usc(c_{i+1} \bowtie p_j)$ and $usc(c_i \bowtie p_j) \geq usc(c_i \bowtie p_{j+1})$
 \Rightarrow if the current “cell” fails, we should *only* examine its *neighbors*, as they are the best possible alternatives.
- Data structure: a priority queue of *candidate cells*
- Its search strategy is reminiscent of the J^* algorithm.

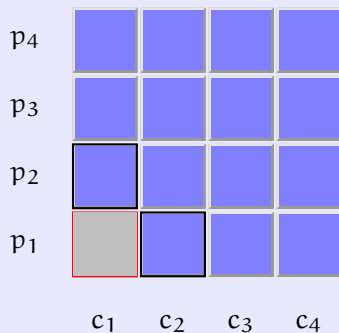


• $c_1 \bowtie p_1$

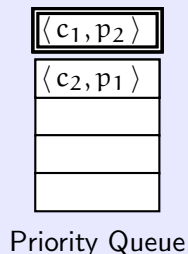


Our Skyline Sweeping Algorithm

- Idea: **be lazy!**
- Based on the **skyline** principle:
 - $usc(c_i \bowtie p_j) \geq usc(c_{i+1} \bowtie p_j)$ and $usc(c_i \bowtie p_j) \geq usc(c_i \bowtie p_{j+1})$
 \Rightarrow if the current “cell” fails, we should *only* examine its *neighbors*, as they are the best possible alternatives.
- Data structure: a priority queue of *candidate cells*
- Its search strategy is reminiscent of the J^* algorithm.

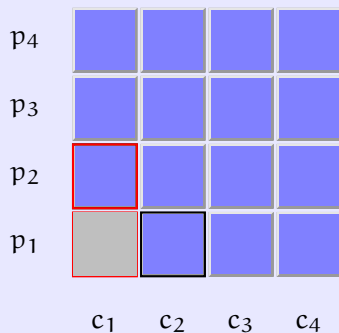


• $c_1 \bowtie p_1$

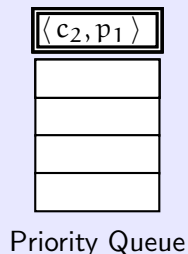


Our Skyline Sweeping Algorithm

- Idea: **be lazy!**
- Based on the **skyline** principle:
 - $usc(c_i \bowtie p_j) \geq usc(c_{i+1} \bowtie p_j)$ and $usc(c_i \bowtie p_j) \geq usc(c_i \bowtie p_{j+1})$
 \Rightarrow if the current “cell” fails, we should *only* examine its *neighbors*, as they are the best possible alternatives.
- Data structure: a priority queue of *candidate cells*
- Its search strategy is reminiscent of the J^* algorithm.

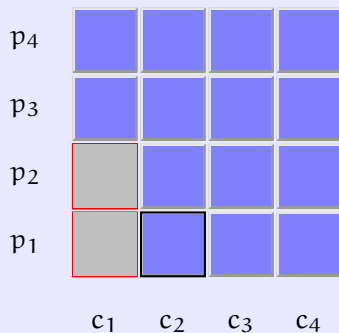


• $c_1 \bowtie p_1$

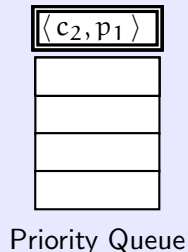


Our Skyline Sweeping Algorithm

- Idea: **be lazy!**
- Based on the **skyline** principle:
 - $usc(c_i \bowtie p_j) \geq usc(c_{i+1} \bowtie p_j)$ and $usc(c_i \bowtie p_j) \geq usc(c_i \bowtie p_{j+1})$
 \Rightarrow if the current “cell” fails, we should *only* examine its *neighbors*, as they are the best possible alternatives.
- Data structure: a priority queue of *candidate cells*
- Its search strategy is reminiscent of the J^* algorithm.

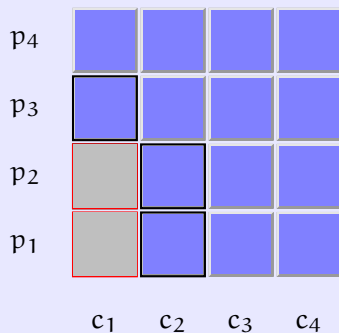


- $c_1 \bowtie p_1$
- $c_1 \bowtie p_2$

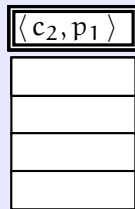


Our Skyline Sweeping Algorithm

- Idea: **be lazy!**
- Based on the **skyline** principle:
 - $usc(c_i \bowtie p_j) \geq usc(c_{i+1} \bowtie p_j)$ and $usc(c_i \bowtie p_j) \geq usc(c_i \bowtie p_{j+1})$
 \Rightarrow if the current “cell” fails, we should *only* examine its *neighbors*, as they are the best possible alternatives.
- Data structure: a priority queue of *candidate cells*
- Its search strategy is reminiscent of the J^* algorithm.



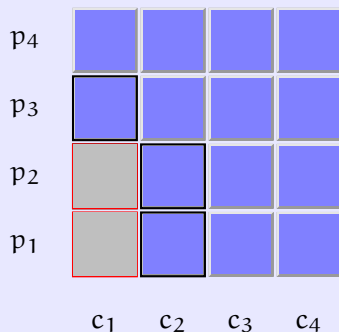
- $c_1 \bowtie p_1$
- $c_1 \bowtie p_2$



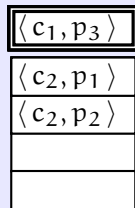
Priority Queue

Our Skyline Sweeping Algorithm

- Idea: **be lazy!**
- Based on the **skyline** principle:
 - $usc(c_i \bowtie p_j) \geq usc(c_{i+1} \bowtie p_j)$ and $usc(c_i \bowtie p_j) \geq usc(c_i \bowtie p_{j+1})$
 \Rightarrow if the current “cell” fails, we should *only* examine its *neighbors*, as they are the best possible alternatives.
- Data structure: a priority queue of *candidate cells*
- Its search strategy is reminiscent of the J^* algorithm.



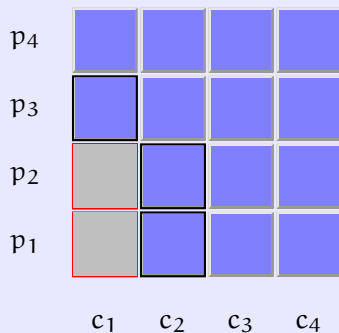
- $c_1 \bowtie p_1$
- $c_1 \bowtie p_2$



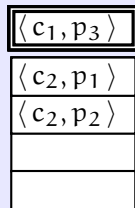
Priority Queue

Our Skyline Sweeping Algorithm

- Idea: **be lazy!**
- Based on the **skyline** principle:
 - $usc(c_i \bowtie p_j) \geq usc(c_{i+1} \bowtie p_j)$ and $usc(c_i \bowtie p_j) \geq usc(c_i \bowtie p_{j+1})$
 \Rightarrow if the current “cell” fails, we should *only* examine its *neighbors*, as they are the best possible alternatives.
- Data structure: a priority queue of *candidate cells*
- Its search strategy is reminiscent of the J^* algorithm.



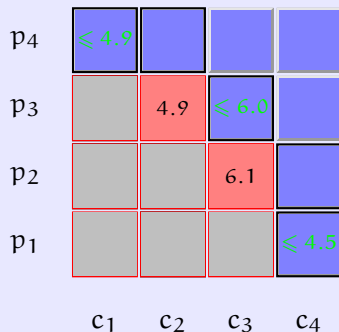
- $c_1 \bowtie p_1$
- $c_1 \bowtie p_2$
- ...



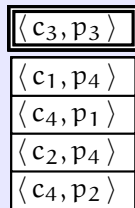
Priority Queue

Our Skyline Sweeping Algorithm

- Idea: **be lazy!**
- Based on the **skyline** principle:
 - $usc(c_i \bowtie p_j) \geq usc(c_{i+1} \bowtie p_j)$ and $usc(c_i \bowtie p_j) \geq usc(c_i \bowtie p_{j+1})$
 \Rightarrow if the current “cell” fails, we should *only* examine its *neighbors*, as they are the best possible alternatives.
- Data structure: a priority queue of *candidate cells*
- Its search strategy is reminiscent of the J^* algorithm.



- $c_1 \bowtie p_1$
- $c_1 \bowtie p_2$
- ...



Priority Queue

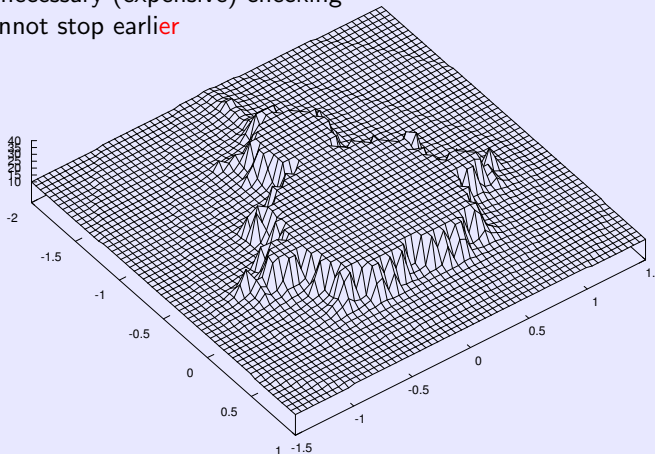
Towards Optimal Probing

- We still pay the price of bounding a non-monotonic function with (a few) monotonic upper bounding functions
 - See an example below
 - Lots of candidates with high $usc(T, Q)$ return much lower (real) score
 - Unnecessary (expensive) checking
 - Cannot stop earlier

Towards Optimal Probing

- We still pay the price of bounding a non-monotonic function with (a few) monotonic upper bounding functions
 - See an example below
 - Lots of candidates with high $usc(T, Q)$ return much lower (real) score
 - Unnecessary (expensive) checking
 - Cannot stop earlier

Mandelbrot function



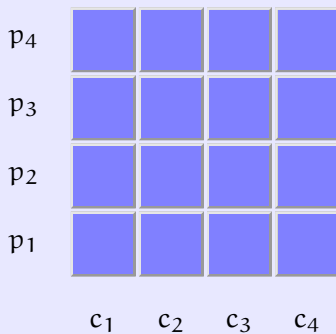
Block Pipeline Algorithm

- Idea: be **even lazier!**
 - Partition the space (into **blocks**) and derive tighter upper bounds for each partitions \Leftarrow “local knowledge”
 - “Unwilling” to check a candidate until we are quite sure about its “prospect” ($\text{bsc}(T, Q)$)
- Details:
 - Group tuples according to their tf signatures — $\langle \text{tf}_{w_1}(t), \dots, \text{tf}_{w_m}(t) \rangle$
 - Form **blocks** — candidate JTTs that has the same tf signatures on all dimensions.
 - The signature of a block b is
$$\text{sig}(b) = \langle \sum_{t \in T} \text{tf}_{w_1}(t), \dots, \sum_{t \in T} \text{tf}_{w_m}(t) \rangle$$
 - A non-monotonic upper bounding function

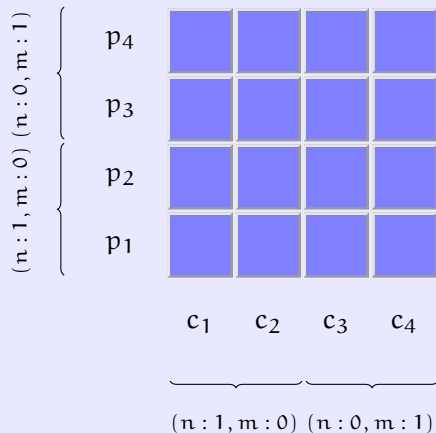
$$\text{bsc}(b, Q) = \sum_{w \in Q \cap b} \frac{1 + \ln(1 + \ln(\text{sig}_w(b)))}{1 - s} \cdot \ln(\text{idf}_w) \cdot \text{sc}_b(b, Q) \cdot \text{sc}_c(\text{CN}(T), Q)$$

- $\text{sc}(t, Q) \leq \text{bsc}(b, Q) \leq \text{usc}(t, Q)$

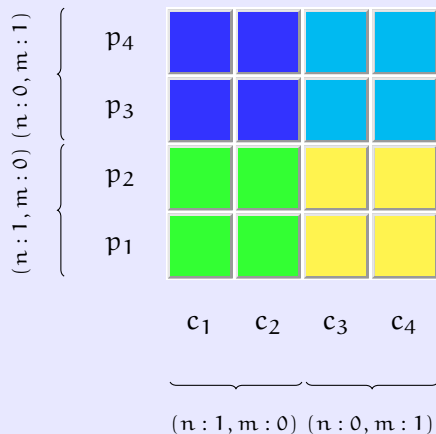
Example of the Block Pipeline Algorithm



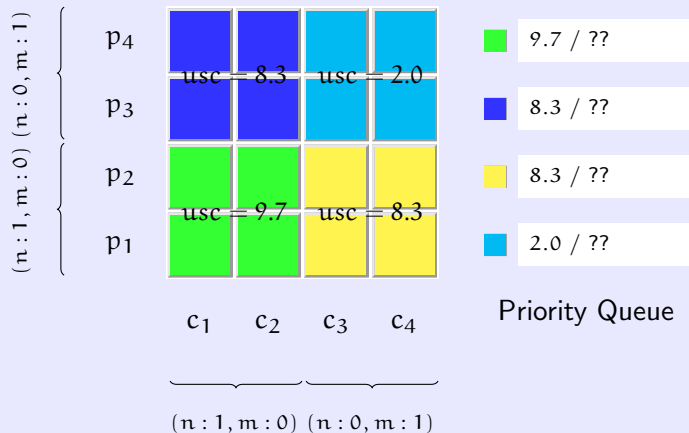
Example of the Block Pipeline Algorithm



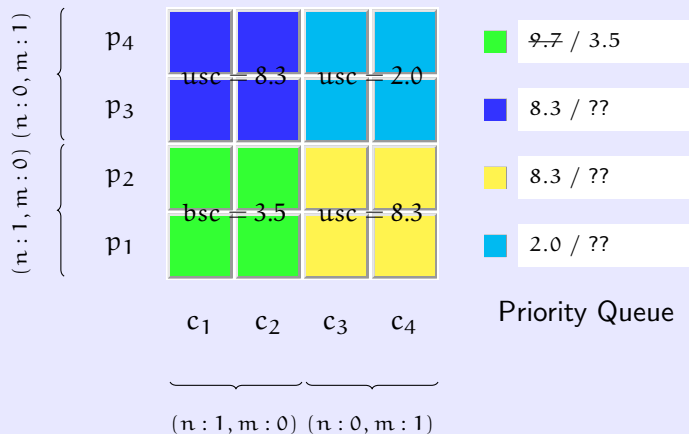
Example of the Block Pipeline Algorithm



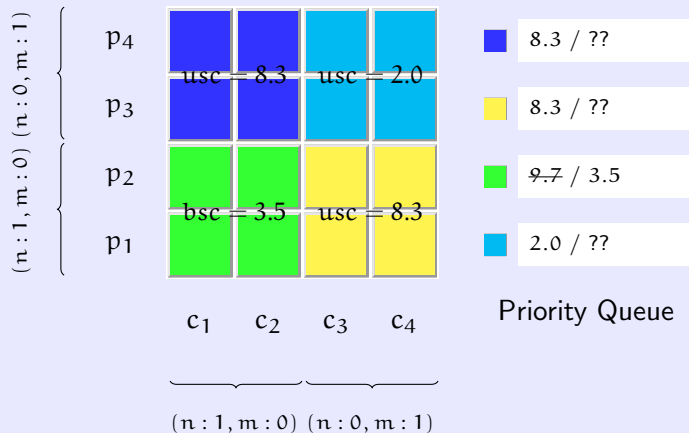
Example of the Block Pipeline Algorithm



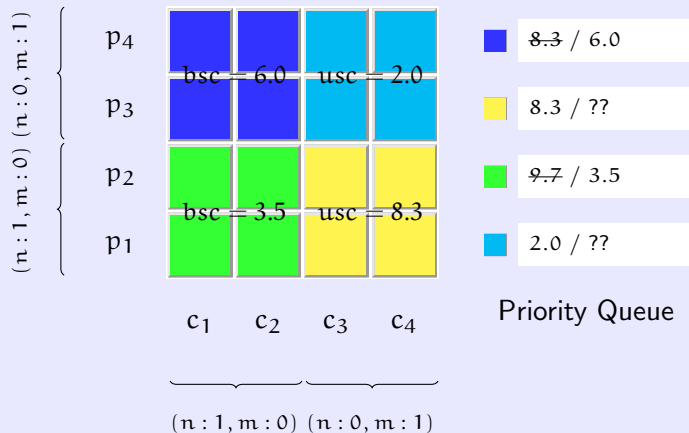
Example of the Block Pipeline Algorithm



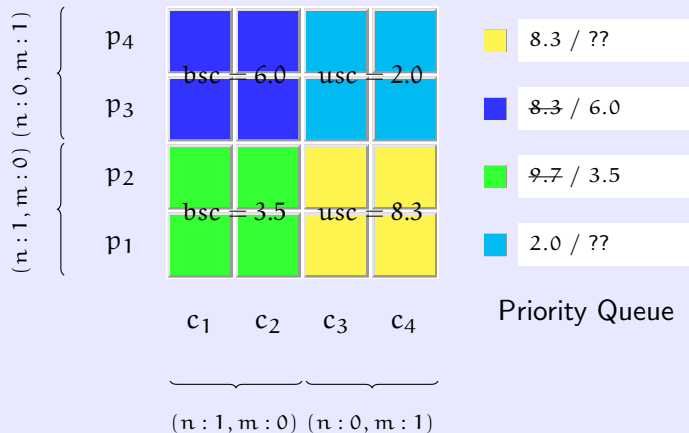
Example of the Block Pipeline Algorithm



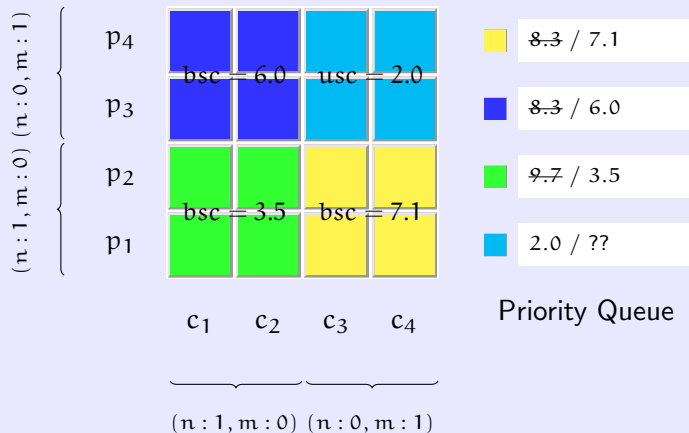
Example of the Block Pipeline Algorithm



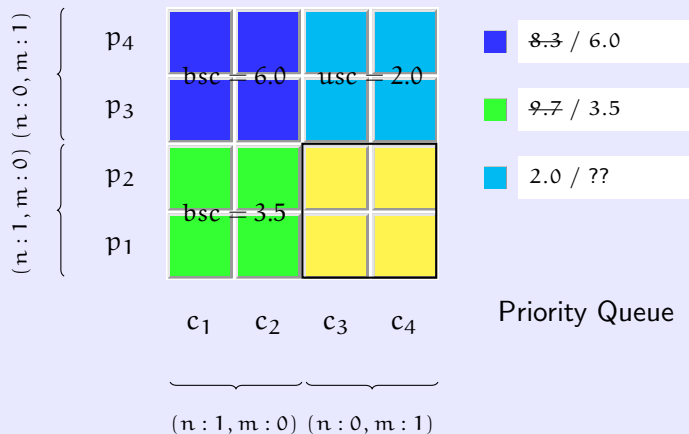
Example of the Block Pipeline Algorithm



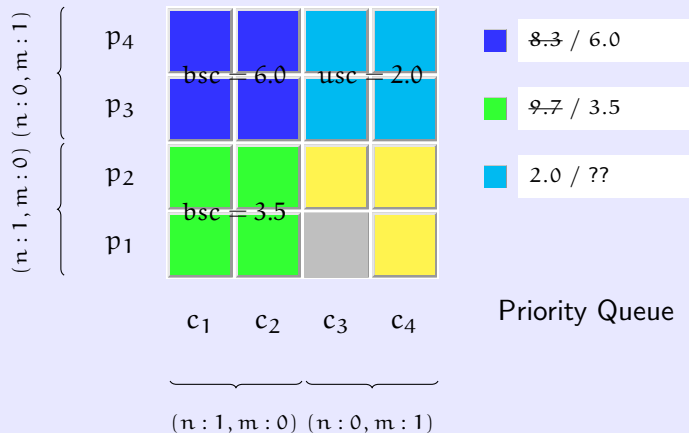
Example of the Block Pipeline Algorithm



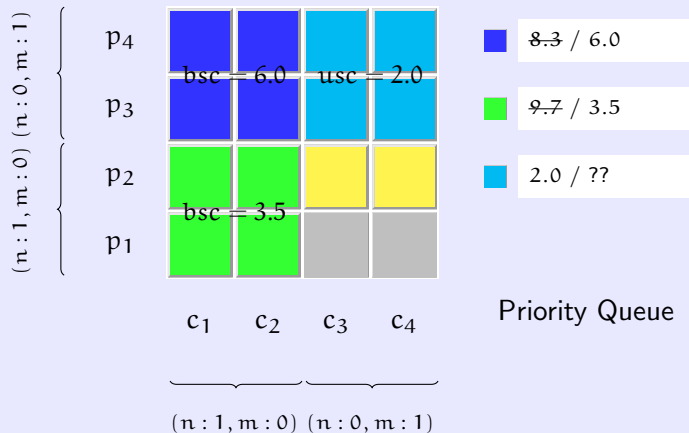
Example of the Block Pipeline Algorithm



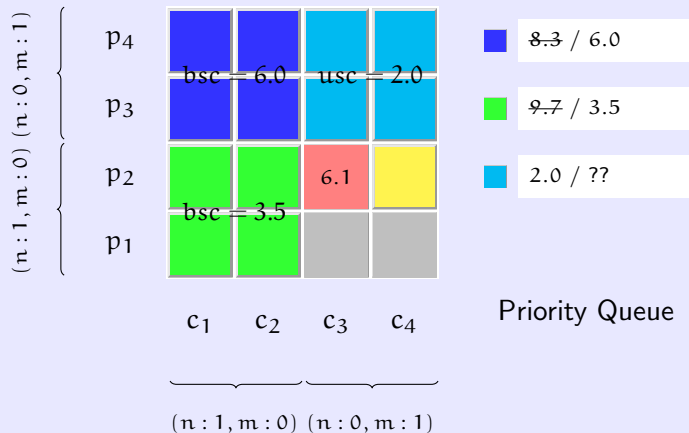
Example of the Block Pipeline Algorithm



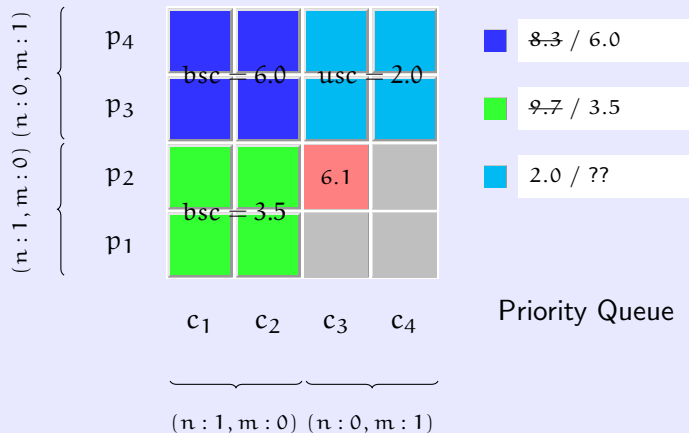
Example of the Block Pipeline Algorithm



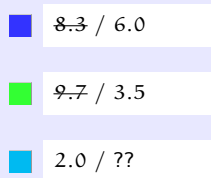
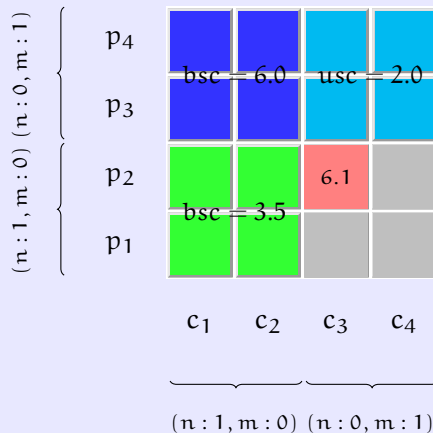
Example of the Block Pipeline Algorithm



Example of the Block Pipeline Algorithm




Example of the Block Pipeline Algorithm



Priority Queue

$$sc(c_3 \bowtie p_2) \geq 6.0 \Rightarrow \text{STOP}$$

- Generalization to Multiple CNs
 - Initially, push lower-left points of each CN into heap, and sort them all together
- Progressively output results
- Optimization: Range Parameterized Join
 - Why?
 - Low join selectivity (i.e., only few candidates really join)
 - High db connection overhead
 - How?
 - Evaluate a block of candidates in one SQL! (e.g., )
 - Group tuples by: tf signature / local score (wtf) / RID

- 1 Introduction
- 2 Ranking Search Results
- 3 Efficient Query Processing
- 4 Experiments**
- 5 Related Work
- 6 Conclusions

Experiment Setup

- Data
 - DBLP: ≈ 0.9 M tuples, 6 tables, 18 queries
 - IMDB: ≈ 10 M tuples, 8 tables, 22 queries
 - Mondial: ≈ 10 K tuples, 28 tables, 35 queries
- Settings
 - **ORACLE** 10g Express / MySQL v5.0.18
 - JDK 1.5 + JDBC
 - 1.8GHz CPU / 512M memory / Debian GNU/Linux 3.1
- Comparison
 - Effectiveness: [VLDB 03], [SIGMOD 06], and ours
 - Efficiency: **Sparse**, **GP** from [VLDB 03], and our **SS** and **BP**
- Metrics
 - Effectiveness
 - **#-Rel**: COUNT(top-1 answer is relevant)
 - **R-Rank**: 1 / (position of the first relevant result)
 - Efficiency
 - Elapsed time
 - ...

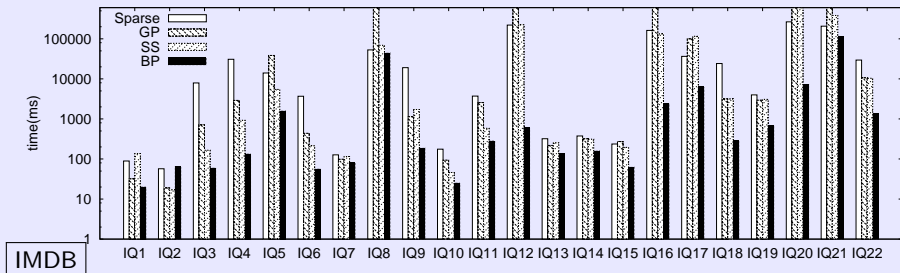
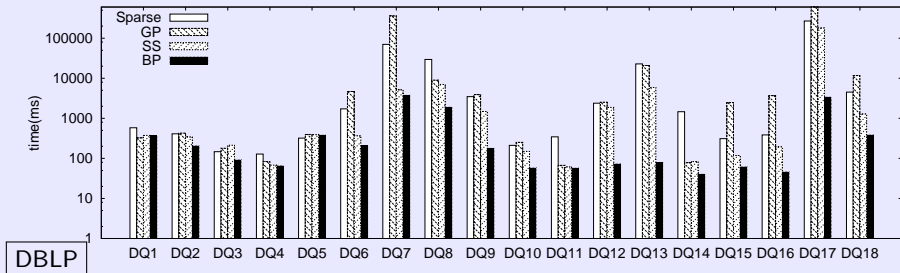
Table: Top-1 Result for Query “nikos clique” on DBLP

Method	Top-1 Result
[Hristidis VLDB03]	InProceeding: <u>Clique-to-Clique</u> Distance ...
[Liu SIGMOD06]	<pre> graph TD S[Series] --> P1[Proceeding] S --> P2[Proceeding] P1 --> R1[Person: Nikos Karatzas] P2 --> R2[InProceeding: Maximum Clique Transversals] P2 --> R3[InProceeding: On ... Clique-Width and ...] </pre>
Ours	Person: <u>Nikos Mamoulis</u> ← RPI → InProceeding: Constraint-Based Algorithms for Computing <u>Clique</u> Intersection Joins

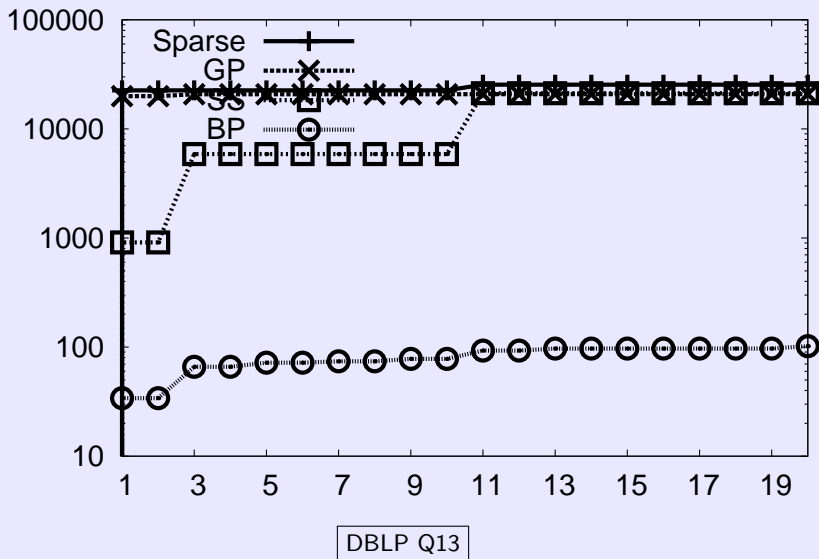
Table: Effectiveness on the DBLP Dataset Based on Top-20 Results

	[Hristidis VLDB03]	[Liu SIGMOD06]	p = 1.0	p = 1.4	p = 2.0
#Rel	2	2	16	16	18
R-Rank	≤ 0.243	≤ 0.333	0.926	0.935	1

Efficiency



Efficiency ...



- 1 Introduction
- 2 Ranking Search Results
- 3 Efficient Query Processing
- 4 Experiments
- 5 Related Work**
- 6 Conclusions

Based on the data model

① Graph-based:

- Proximity Search [VLDB 98]
- BANKS I [ICDE 02], BANKS II [VLDB 05]
- Recent work [PODS 06, ICDE 07, SIGMOD 07]

Strength Fast query response

Weakness Main-memory based; hard to integrate IR-style ranking functions

② Relation-based:

- DBXplorer [VLDB 03]
- DISCOVER I [VLDB 02], DISCOVER II [VLDB 03]
- Recent work [IDEAS 05, SIGMOD 06]

Strength Ease of maintenance and deployment; allow sophisticated ranking

Weakness Query response time

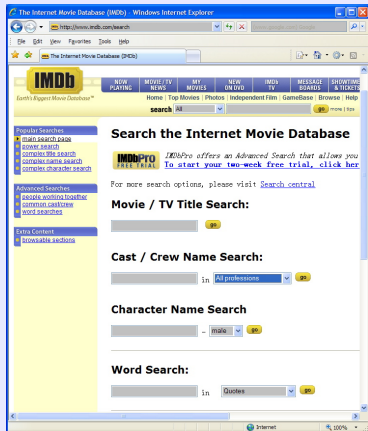
- 1 Introduction
- 2 Ranking Search Results
- 3 Efficient Query Processing
- 4 Experiments
- 5 Related Work
- 6 Conclusions**

Conclusions

- Say “no” to SQL, say “Yes” to keyword search.

Conclusions

- Say “no” to SQL, say “Yes” to keyword search.



OR

Conclusions

- Say “no” to SQL, say “Yes” to keyword search.

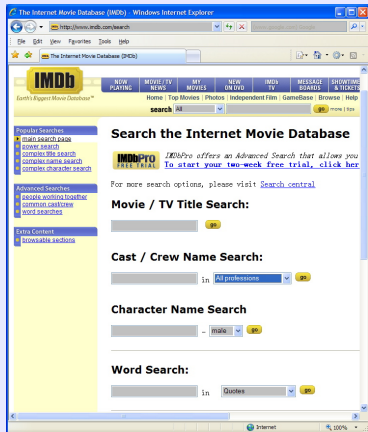


OR



Conclusions

- Say “no” to SQL, say “Yes” to keyword search.
 - It is effective: Meaningful query results with appropriate rankings



OR



Conclusions

- Say “no” to SQL, say “Yes” to keyword search.
 - It is effective: Meaningful query results with appropriate rankings
 - It is efficient: Second-level response time for $\approx 10M$ tuple DB on a commodity PC



OR



Thank You!

Query Results - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

Mondial DBLP IMDb Northwind

2001 Hanks Search! Advanced Search

IMDb Results 1 - 10 in " 40.54 for " 2001 Hanks "

movies : NAME: "Primetime Glick" (2001) { Tom Hanks/Ben Stiller (#2.1)} 31.54

actors : NAME: Hanks, Tom 19.77

actorplay : CHARACTER: Himself

movies : NAME: "Primetime Glick" (2001) { Tom Hanks/Ben Stiller (#2.1)}

actors : NAME: Hanks, Colin 17.87

actorplay : CHARACTER: Alex Whitman (1999-2001)

actorplay : CHARACTER: John Hanks 17.87

movies : NAME: Rosamunde Pilcher - Wind über dem Fluss (2001) (TV)