Interval Count Semi-Joins

Panagiotis Bouros[1] and Nikos Mamoulis[2]
[1]Institute of Computer Science, Johannes Gutenberg University Mainz, Germany
[2]Department of Computer Science & Engineering, University of Ioannina, Greece
bouros@uni-mainz.de, nikos@cs.uoi.gr

Introduction

Company R

employee	start	end
John	1993	2003
Mary	1995	2008
Tom	2000	2015

Company S

employee	start	end
Jane	1994	1996
Bob	1998	2002
Hugo	2007	2010
Helen	2014	2017

Interval Join

Find all pairs of employees whose working periods at companies **R** and **S** *overlap*

Interval Count Semi-Join

For *each* employee *r* from company **R**, count how many employees from company **S** worked at a period of time *overlapping* with *r*'s employment

Applications

- *Temporal* databases
- *Selecting and/or ranking* objects

Related work

- *Plane-sweep* based interval joins [1], [2]
- *Top-k count semi-joins*
  - Relational [3], spatial [4]

Background

Interval Join based on plane sweep

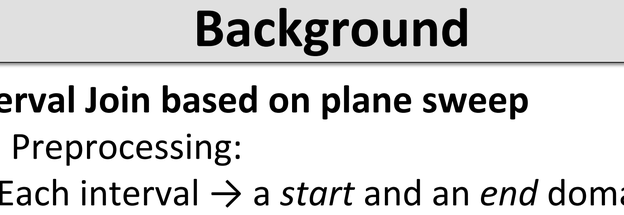
① Preprocessing:

- Each interval → a *start* and an *end* domain point
- Sort endpoints of each input collection

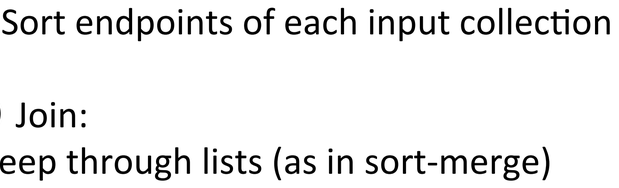
② Join:

Sweep through lists (as in sort-merge)

- If a *start* domain point (e.g. from **R**) is met, insert interval to *active set* (e.g. A^R) scan other active set (e.g. A^S) to produce join results
- If an *end* domain point (e.g. from **R**) is met, delete interval from active set (e.g. A^R)



A^R={John, Mary}, results+={(John, Bob), (Mary, Bob)}

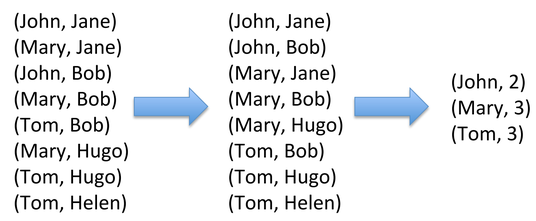


A^S={Bob}, results+={(Tom, Bob)}

Baseline Approaches

Naïve Algorithm:

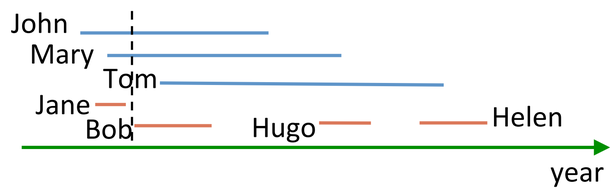
- Compute interval join
- Sort and aggregate join results



✗ High cost: join + aggregation

Simple Counting Algorithm:

- Adapt Interval Join algorithm to count ICSJ results instead of producing IJ results



A^R={John, Mary}, John.*count*+=1, Mary.*count*+=1

A^S={Bob}, Tom.count+=1

✗ Cost similar to interval join

✗ Cost sensitive to join output

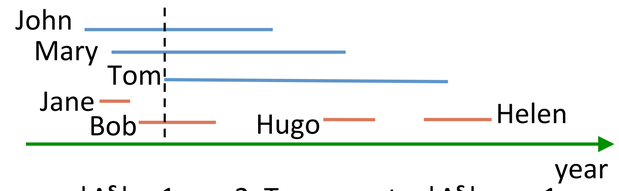
Smart Counting Algorithm

Observation: to compute an *r.count* we need:

- The number $|A^S|$ of active intervals from **S** when *r.start* is accessed
- The number of intervals form **S** which became active after *r.start* and before *r.end*

Approach: use a *global counter g* to track for how many intervals $s \in S$ their start point has been seen so far

- When the *start* point of a $s \in S$ is accessed → increase $|A^S|$; increase *g*
- When the *end* point of a $s \in S$ is accessed → decrease $|A^S|$
- When the *start* point of a $r \in R$ is accessed → initialize *r.count* = $|A^S|$ - *g*
- When the *end* point of a $r \in R$ is accessed → finalize *r.count* = *r.count* + *g*



$|A^S|$ = 1, *g* = 2, Tom.*count* = $|A^S|$-*g* = -1

$|A^S|$ = 1, *g* = 4, Tom.*count* += *g*, output(Tom.*count*, 3)

- Use a hash table (or array) to track *r.count* 's
- ✓ Complexity: O(|**R**|+|**S**|), excluding sorting

Experiments

Setup

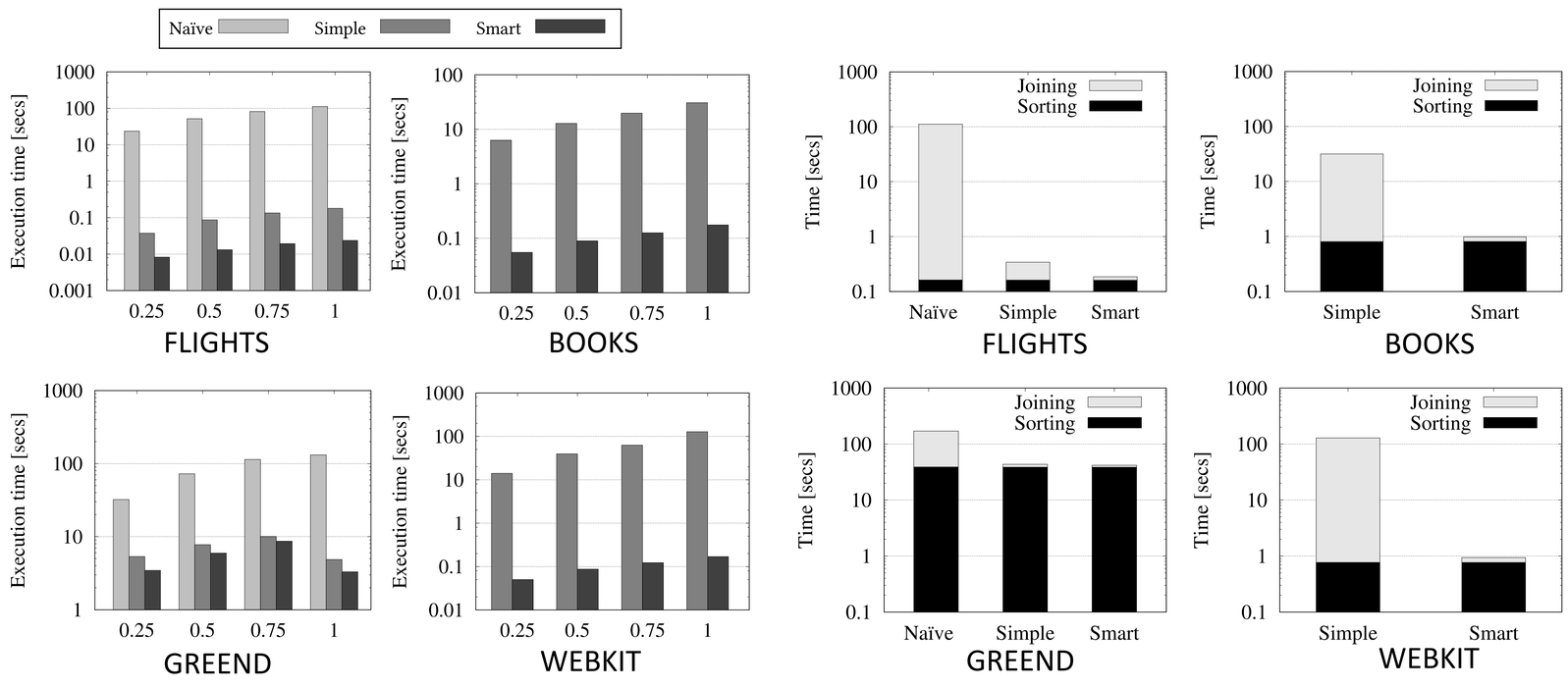
- In-memory processing
- *Gapless hash map* [2]
- *Lazy optimization* [2] for Naïve and Simple Counting

Datasets

	FLIGHTS	BOOKS	GREEND	WEBKIT
Cardinality	445,827	2,312,602	110,115,441	2,347,346
Domain duration (secs)	2,750,280	31,507,200	283,356,410	461,829,284
Shortest interval (secs)	1,261	1	1	1
Longest interval (secs)	42,301	31,406,400	59,468,008	461,815,512
Avg. interval duration (secs)	8,791	2,201,320	16	33,206,300
Distinct domain points	41,975	5,330	182,028,123	174,471

Execution time (varying |R|/|S|)

Breakdown (|R|=|S|)



References

[1] P. Bouros and N. Mamoulis. A Forward Scan based Plane Sweep Algorithm for Parallel Interval Joins. In PVLDB, 10(11), 2017.
[2] D. Piatov, S. Helmer, and A. Dignös. An interval join optimized for modern hardware. In ICDE, 2016.
[3] C. Sheng, Y. Tao, and J. Li. Exact and approximate algorithms for the most connected vertex problem. ACM TODS 37(2), 2012.
[4] M. Zhu, D. Papadias, J. Zhang, and D. L. Lee. Top-k Spatial Joins. IEEE TKDE 17(4), 2005.

21st International Conference on Extending Database Technology (EDBT), Vienna, Austria, March 26-29, 2018