Efficient Top-*k* Spatial Distance Joins

Shuyao Qi[1]

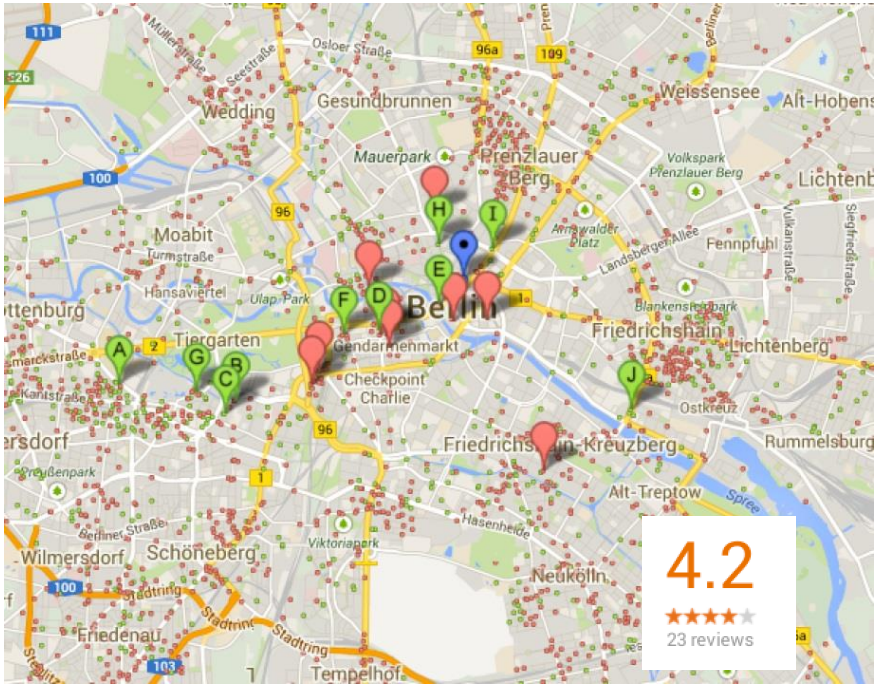
Panagiotis Bouros[1,2]

Nikos Mamoulis[1]

[1]The University of Hong Kong
[2]Humboldt-Universität zu Berlin

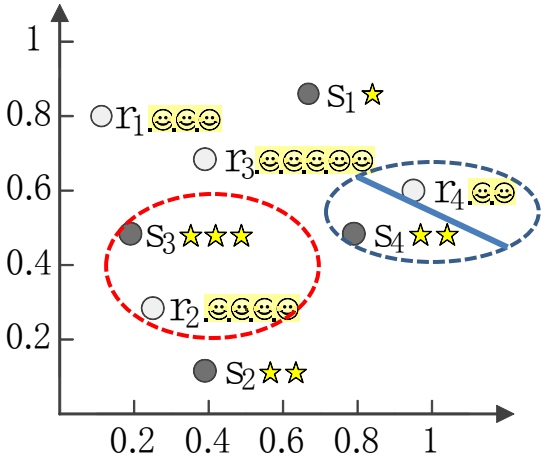
Motivation Example

- Observation
  - More spatial locations are tagged with score information (quality, rating, price, etc)
- Spatial object
  - Location: (*x*, *y*)
  - Score



Motivation Example

- Dataset *R*: restaurants
- Dataset *S*: hotels
- Recommend to tourists hotel-restaurant pairs, which are
  - (1) close to each other (spatial join)
  - (2) of high quality (top-*k* selection)



Example

$k = 1, \epsilon = 0.2, SUM$

Result

(r_2, s_3)

Definition

- Top-*k* Spatial Distance Joins (*k*-SDJ)
  - Two collections of spatial objects *R* and *S* (*id, loc, score*)
  - The *k*-SDJ query retrieves a k-subset of $R \times S$ such that for every result pair (r, s):
    - (1) r is spatially close to s (i.e., $dist(r, s) \leq \epsilon$)
    - (2) it has top-*k* largest aggregation score $\gamma(r, s)$
      - γ is a monotone function

Contributions

- k-SDJ query
  - Location-based services
  - data integration
  - geoscience, etc.
- Algorithms
  - SFA, DFA, BA
- Experiments
  - Effectiveness and efficiency

Outline

- Related Work
  - Top-k Join Queries
  - Spatial Join Queries
- Algorithms
- Experiments
- Conclusions

Top-*k* Join Queries

- Rank aggregation on top of relational join results, i.e., assume simple join attributes and join predicate [Ilyas 2008]
- Hash-Based Rank-Join (HRJN) [Ilyas 2003]
  - Sort *R* and *S* descending to scores
  - HashTable indexes on *R* and *S*
  - Each accessed record is joined with the opposite hash table

Spatial Join Queries

- ϵ-Distance Join
  - Given two sets of spatial objects R and S, identify the object pairs (r, s) with $r \in R, s \in S$, such that $dist(r, s) \leq \epsilon$
  - PlaneSweep Algorithm [Arge 1998]
    - For non-indexed inputs, sort and sweep
  - R-tree Join Algorithm [Brinkhoff 1993]
    - For inputs indexed by R-trees

Outline

- Related Work
- Algorithms
- Experiments
- Conclusions

k-SDJ Evaluation

- Combination of top-*k* query and spatial join
  - Score: $\gamma(r, s)$
  - Distance: $dist(r, s)$
- Challenges
  - Effective pruning and termination techniques based on both dimensions
- Ideas
  - Index both spatial and score attributes
  - Score-first/Distance-first/Hybrid processing

Indexing: aR-tree

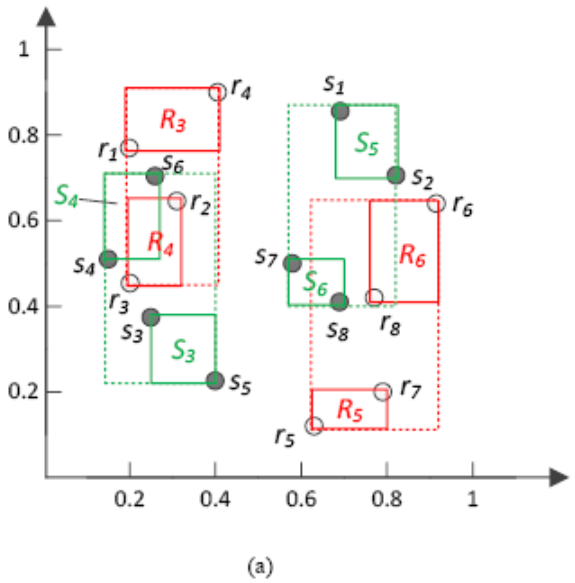
- R-tree augmented with maximum score information
- Property:
  - Given two aR-tree entries e_R and e_S. If $\gamma(e_R, e_S) = \beta, \forall r_i \in e_R, \forall s_j \in e_S$, $\gamma(r_i, s_j) \leq \beta$

id	*loc*	*score*
r_1	(0.20, 0.78)	1.0
r_2	(0.30, 0.64)	0.8
r_3	(0.20, 0.45)	0.8
r_4	(0.40, 0.90)	0.6
r_5	(0.63, 0.12)	0.6
r_6	(0.91, 0.63)	0.4
r_7	(0.79, 0.20)	0.3
r_8	(0.76, 0.42)	0.1

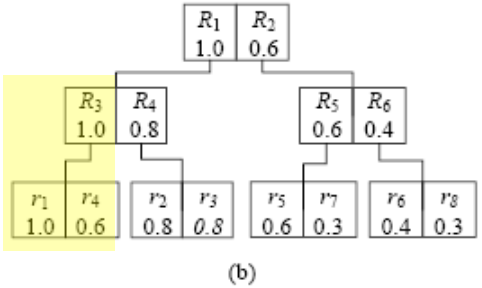
Object collection R

id	*loc*	*score*
s_1	(0.69, 0.85)	0.9
s_2	(0.81, 0.71)	0.9
s_3	(0.24, 0.38)	0.8
s_4	(0.15, 0.52)	0.7
s_5	(0.40, 0.22)	0.7
s_6	(0.25, 0.70)	0.4
s_7	(0.58, 0.50)	0.4
s_8	(0.68, 0.42)	0.2

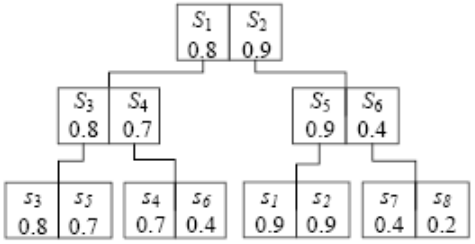
Object collection S



(a)



(b)



(c)

Score-First Algorithm (SFA)

- HRJN variant with aR-trees
  - Sorted according to scores
  - aR-tree indexes A_R and A_S on R/S
  - Each accessed record is
    - (1) Joined with the opposite aR-tree
    - (2) Inserted into corresponding aR-tree
  - Termination based on score bounds

Score-First Algorithm (SFA)

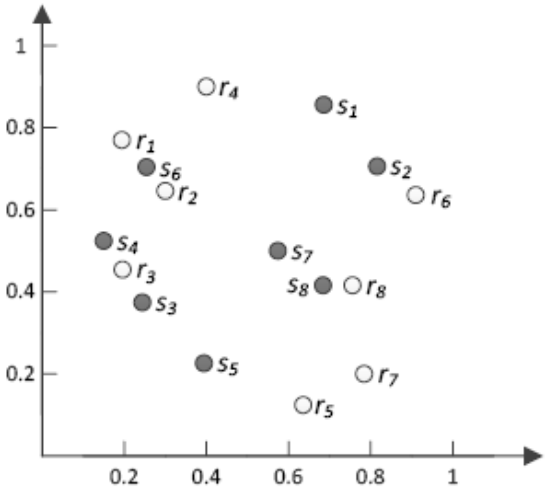
- Example

id	loc	$score$
r_1	(0.20, 0.78)	1.0
r_2	(0.30, 0.64)	0.8
r_3	(0.20, 0.45)	0.8
r_4	(0.40, 0.90)	0.6
r_5	(0.63, 0.12)	0.6
r_6	(0.91, 0.63)	0.4
r_7	(0.79, 0.20)	0.3
r_8	(0.76, 0.42)	0.1

Object collection R

id	loc	$score$
s_1	(0.69, 0.85)	0.9
s_2	(0.81, 0.71)	0.9
s_3	(0.24, 0.38)	0.8
s_4	(0.15, 0.52)	0.7
s_5	(0.40, 0.22)	0.7
s_6	(0.25, 0.70)	0.4
s_7	(0.58, 0.50)	0.4
s_8	(0.68, 0.42)	0.2

Object collection S



SFA Example:

$k = 1, \epsilon = 0.1, \gamma = SUM$

Round 1:

$A_R = \{r_1\}$

$A_S = \{\}$

Score-First Algorithm (SFA)

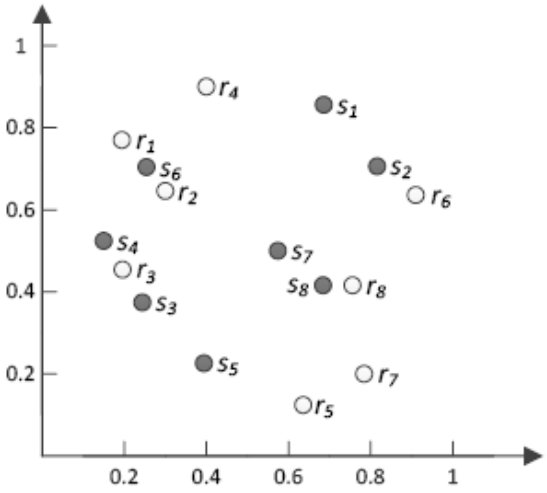
- Example

id	*loc*	*score*
r_1	(0.20, 0.78)	1.0
r_2	(0.30, 0.64)	0.8
r_3	(0.20, 0.45)	0.8
r_4	(0.40, 0.90)	0.6
r_5	(0.63, 0.12)	0.6
r_6	(0.91, 0.63)	0.4
r_7	(0.79, 0.20)	0.3
r_8	(0.76, 0.42)	0.1

Object collection R

id	*loc*	*score*
s_1	(0.69, 0.85)	0.9
s_2	(0.81, 0.71)	0.9
s_3	(0.24, 0.38)	0.8
s_4	(0.15, 0.52)	0.7
s_5	(0.40, 0.22)	0.7
s_6	(0.25, 0.70)	0.4
s_7	(0.58, 0.50)	0.4
s_8	(0.68, 0.42)	0.2

Object collection S



SFA Example:

$k = 1, \epsilon = 0.1, \gamma = SUM$

Round 2:

$A_R = \{r_1\}$

$A_S = \{s_1\}$

Score-First Algorithm (SFA)

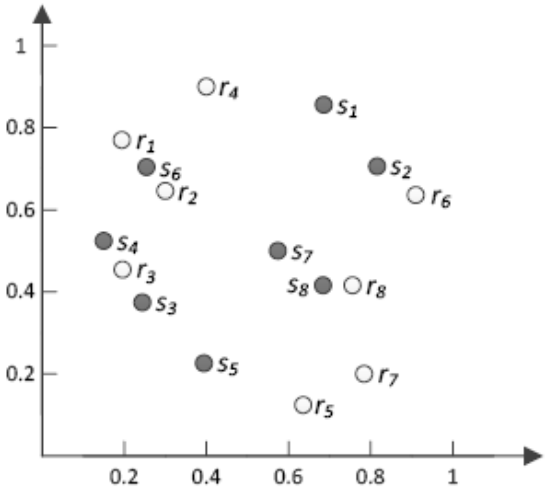
- Example

id	*loc*	*score*
r_1	(0.20, 0.78)	1.0
r_2	(0.30, 0.64)	0.8
r_3	(0.20, 0.45)	0.8
r_4	(0.40, 0.90)	0.6
r_5	(0.63, 0.12)	0.6
r_6	(0.91, 0.63)	0.4
r_7	(0.79, 0.20)	0.3
r_8	(0.76, 0.42)	0.1

Object collection R

id	*loc*	*score*
s_1	(0.69, 0.85)	0.9
s_2	(0.81, 0.71)	0.9
s_3	(0.24, 0.38)	0.8
s_4	(0.15, 0.52)	0.7
s_5	(0.40, 0.22)	0.7
s_6	(0.25, 0.70)	0.4
s_7	(0.58, 0.50)	0.4
s_8	(0.68, 0.42)	0.2

Object collection S



SFA Example:

$k = 1, \epsilon = 0.1, \gamma = SUM$

Round 3:

$A_R = \{r_1, r_2\}$

$A_S = \{s_1\}$

Score-First Algorithm (SFA)

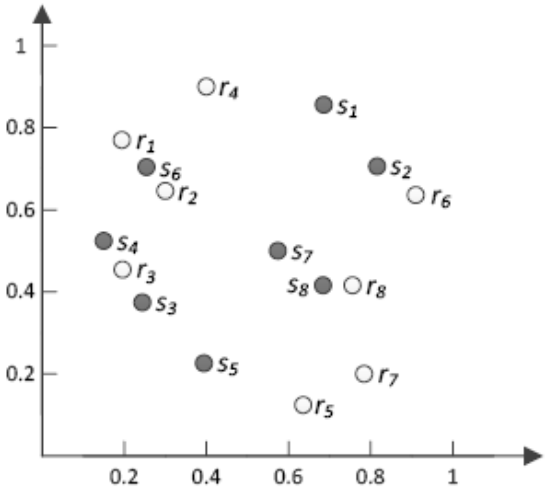
- Example

id	*loc*	*score*
r_1	(0.20, 0.78)	1.0
r_2	(0.30, 0.64)	0.8
r_3	(0.20, 0.45)	0.8
r_4	(0.40, 0.90)	0.6
r_5	(0.63, 0.12)	0.6
r_6	(0.91, 0.63)	0.4
r_7	(0.79, 0.20)	0.3
r_8	(0.76, 0.42)	0.1

Object collection R

id	*loc*	*score*
s_1	(0.69, 0.85)	0.9
s_2	(0.81, 0.71)	0.9
s_3	(0.24, 0.38)	0.8
s_4	(0.15, 0.52)	0.7
s_5	(0.40, 0.22)	0.7
s_6	(0.25, 0.70)	0.4
s_7	(0.58, 0.50)	0.4
s_8	(0.68, 0.42)	0.2

Object collection S



SFA Example:

$k = 1, \epsilon = 0.1, \gamma = SUM$

Round 4:

$A_R = \{r_1, r_2\}$

$A_S = \{s_1, s_2\}$

Score-First Algorithm (SFA)

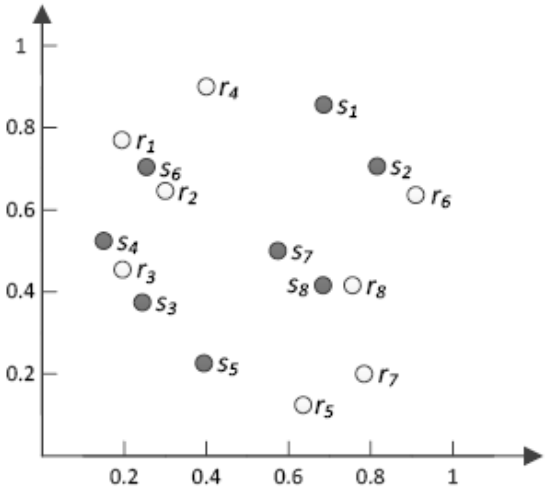
- Example

id	*loc*	*score*
r_1	(0.20, 0.78)	1.0
r_2	(0.30, 0.64)	0.8
r_3	(0.20, 0.45)	0.8
r_4	(0.40, 0.90)	0.6
r_5	(0.63, 0.12)	0.6
r_6	(0.91, 0.63)	0.4
r_7	(0.79, 0.20)	0.3
r_8	(0.76, 0.42)	0.1

Object collection R

id	*loc*	*score*
s_1	(0.69, 0.85)	0.9
s_2	(0.81, 0.71)	0.9
s_3	(0.24, 0.38)	0.8
s_4	(0.15, 0.52)	0.7
s_5	(0.40, 0.22)	0.7
s_6	(0.25, 0.70)	0.4
s_7	(0.58, 0.50)	0.4
s_8	(0.68, 0.42)	0.2

Object collection S



SFA Example:

$k = 1, \epsilon = 0.1, \gamma = SUM$

Round 6:

$A_R = \{r_1, r_2, s_3\}$

$A_S = \{s_1, s_2, s_3\}$

$C = \{(r_3, s_3) = 1.6\}$

Upper bound of remaining records: 1.8

Score-First Algorithm (SFA)

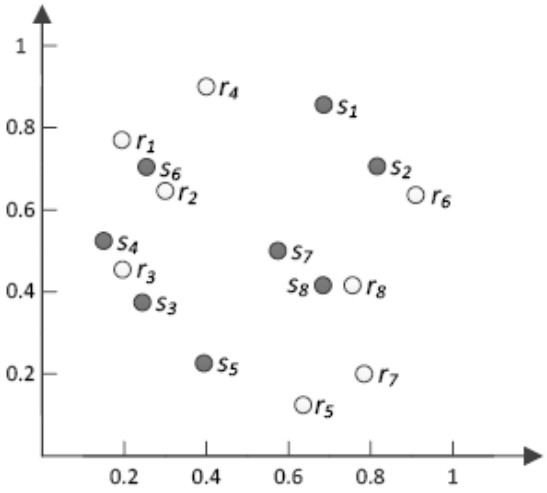
- Example

id	*loc*	*score*
r_1	(0.20, 0.78)	1.0
r_2	(0.30, 0.64)	0.8
r_3	(0.20, 0.45)	0.8
r_4	(0.40, 0.90)	0.6
r_5	(0.63, 0.12)	0.6
r_6	(0.91, 0.63)	0.4
r_7	(0.79, 0.20)	0.3
r_8	(0.76, 0.42)	0.1

Object collection R

id	*loc*	*score*
s_1	(0.69, 0.85)	0.9
s_2	(0.81, 0.71)	0.9
s_3	(0.24, 0.38)	0.8
s_4	(0.15, 0.52)	0.7
s_5	(0.40, 0.22)	0.7
s_6	(0.25, 0.70)	0.4
s_7	(0.58, 0.50)	0.4
s_8	(0.68, 0.42)	0.2

Object collection S



SFA Example:

$k = 1, \epsilon = 0.1, \gamma = SUM$

Round 10:

$A_R = \{r_1, r_2, r_3, r_4\}$

$A_S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$

$C = \{(r_3, s_3) = 1.6\}$

Upper bound of remaining records: *1.5*

return C

Score-First Algorithm (SFA)

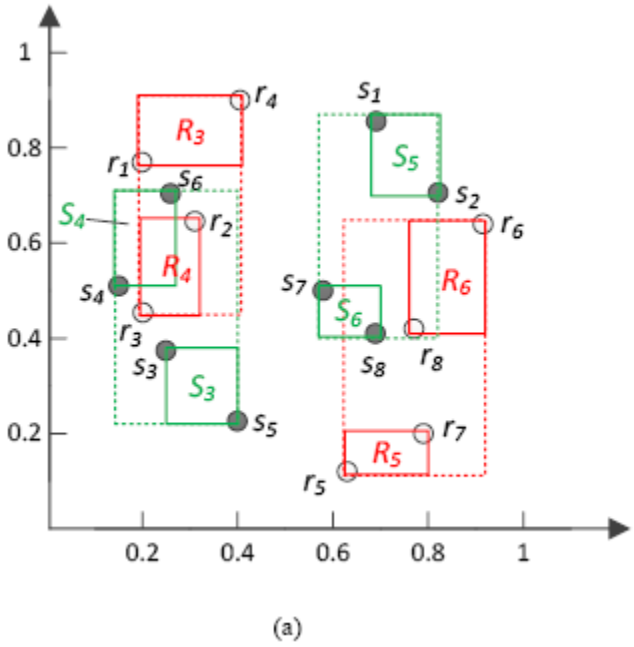
- Drawbacks:
  - Update operation on aR-trees
  - Range query operation for each record
  - Inefficient in spatial join computation

Distance-First Algorithm (DFA)

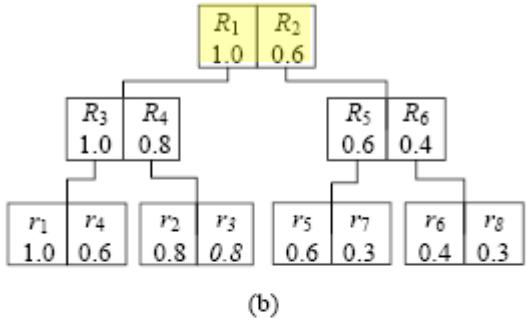
- General idea: Join -> Sort
- Challenge: Progressive join based on both spatial and aggregate score information
  - Bulk-load R and S to build aR-tree indexes A_R and A_S
  - Best-first order traversal
    - Use max-heap to prioritize aR-tree entry pairs based on $\gamma(e_R, e_S)$
  - Rationale: Entries containing objects with larger scores are processed first and thus enabling early termination

Distance-First Algorithm (DFA)

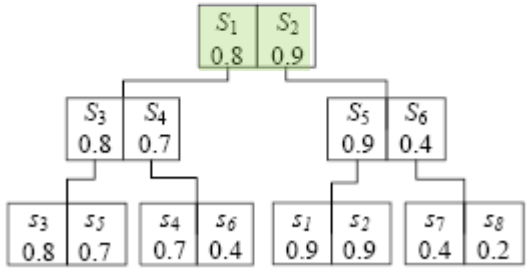
- Example



(a)



(b)



(c)

DFA Example:

$k = 1, \epsilon = 0.1, \gamma = SUM$

Round 1:

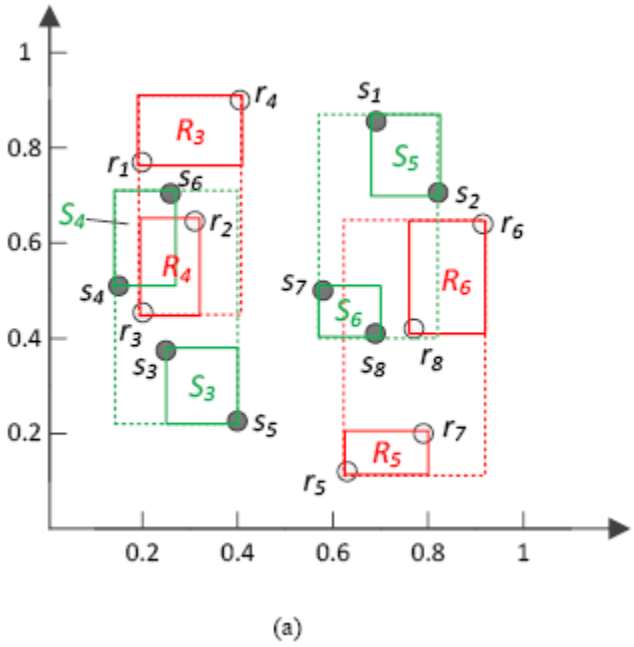
$R_{root} \bowtie S_{root} = \{R_1S_1, R_2S_2\}$

H_e
$(R_1, S_1), 1.8$
$(R_2, S_2), 1.5$

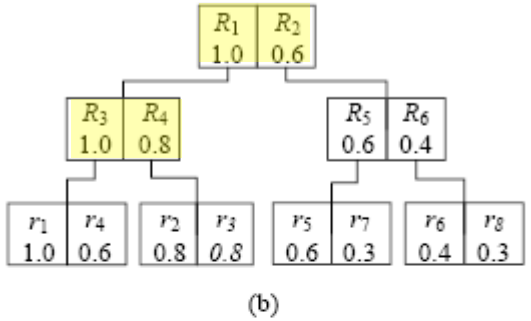
R_1S_2, R_2S_1:
pruned by distance

Distance-First Algorithm (DFA)

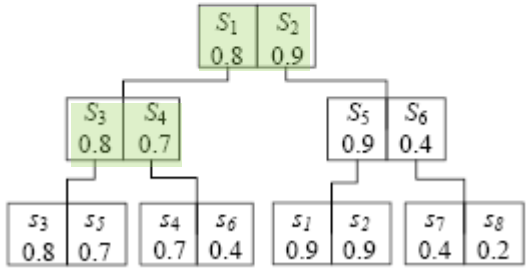
- Example



(a)



(b)



(c)

DFA Example:

$k = 1, \epsilon = 0.1, \gamma = SUM$

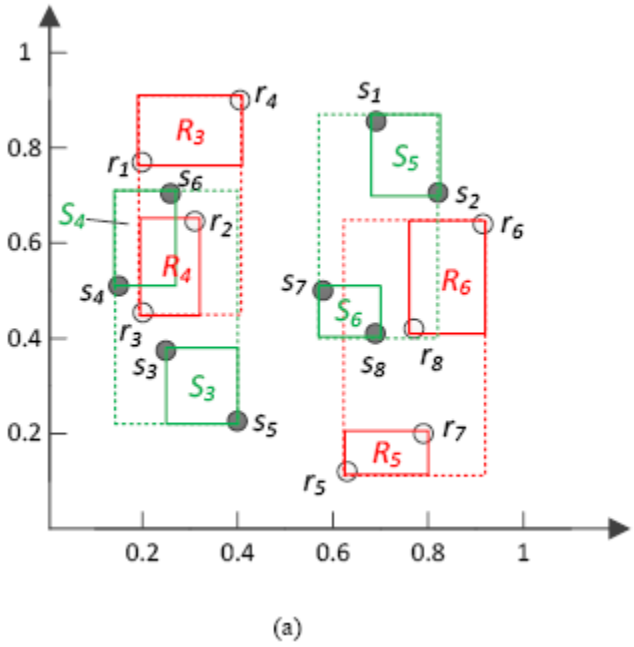
Round 2:

$R_1 \bowtie S_1 = \{R_3S_4, R_4S_3, R_4S_4\}$

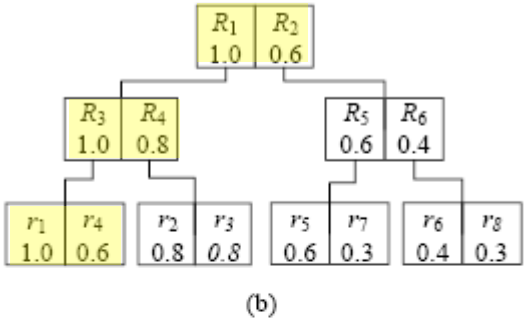
$\boldsymbol{H_e}$
$(R_3, S_4), 1.7$
$(R_4, S_3), 1.6$
$(R_4, S_4), 1.5$
$(R_2, S_2), 1.5$

Distance-First Algorithm (DFA)

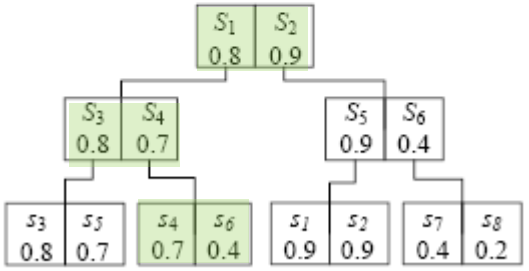
- Example



(a)



(b)



(c)

DFA Example:

$k = 1, \epsilon = 0.1, \gamma = SUM$

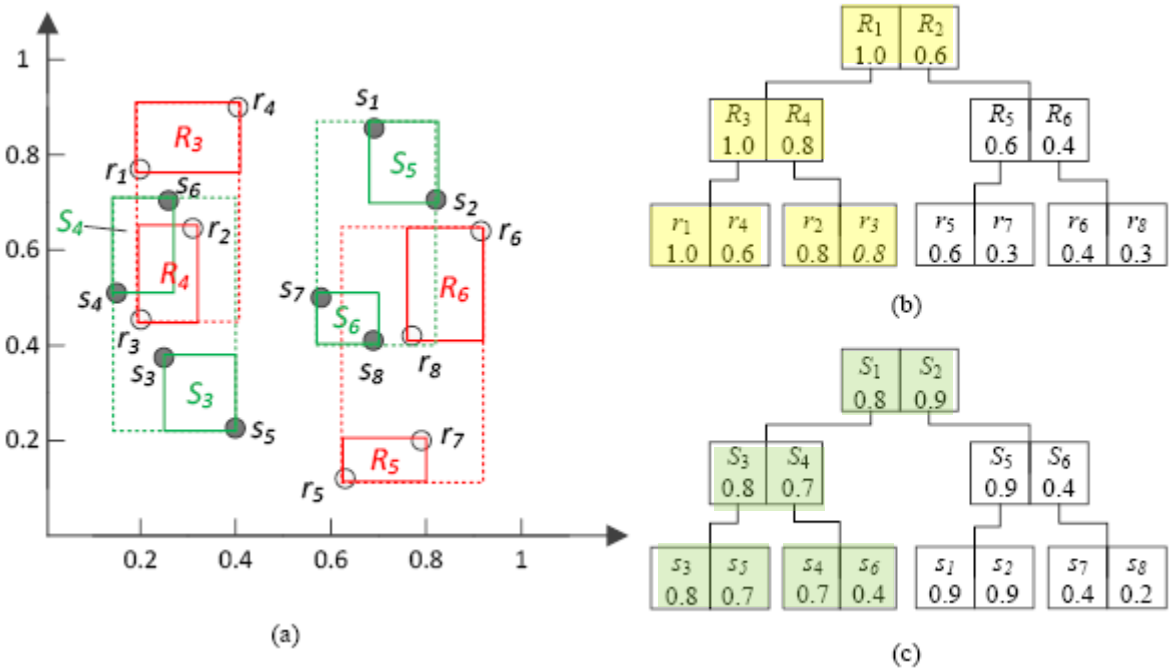
Round 3:

$R_3 \bowtie S_4 = \{r_1 s_6\}$

H_e
$(R_4, S_3), 1.6$
$(R_4, S_4), 1.5$
$(R_2, S_2), 1.5$
$(r_1, s_6), 1.4$

Distance-First Algorithm (DFA)

- Example



DFA Example:
$k = 1, \epsilon = 0.1, \gamma = SUM$

Round 3:
$R_4 \bowtie S_3 = \{r_3 s_3\}$

H_e
$(r_3, s_3), 1.6$
$(R_4, S_4), 1.5$
$(R_2, S_2), 1.5$
$(r_1, s_6), 1.4$

Given two aR-tree entries e_R and e_S. If $\gamma(e_R, e_S) = \beta, \forall r_i \in e_R, \forall s_j \in e_S, \gamma(r_i, s_j) \leq \beta$

$C = \{(r_3, s_3)\}$
return C

SFA vs. DFA

	SFA	DFA
Dataset	Score-ordered partial traversal	Index the whole datasets
Index	Update the aR-trees for each record	Bulk-load to build the aR-trees
Join	Probe the index for each record	Tree level join

Block-based Algorithm (BA)

- Challenge: exploit the benefits of SFA and DFA, while avoid their disadvantages
- Idea: Retrieve sorted records in block-wise fashion, bulk-load and join small blocks
  - Retrieve records in order of scores, thus able to terminate like SFA
  - Bulk-load aR-tree on each block for only once
  - Perform DFA on block level

Block-based Algorithm (BA)

- Example

	id	*loc*	*score*
b_{R1}	r_1	(0.20, 0.78)	1.0
	r_2	(0.30, 0.64)	0.8
b_{R2}	r_3	(0.20, 0.45)	0.8
	r_4	(0.40, 0.90)	0.6
b_{R3}	r_5	(0.63, 0.12)	0.6
	r_6	(0.91, 0.63)	0.4
b_{R4}	r_7	(0.79, 0.20)	0.3
	r_8	(0.76, 0.42)	0.1

(a) collection R

	id	*loc*	*score*
b_{S1}	s_1	(0.69, 0.85)	0.9
	s_2	(0.81, 0.71)	0.9
b_{S2}	s_3	(0.24, 0.38)	0.8
	s_4	(0.15, 0.52)	0.7
b_{S3}	s_5	(0.40, 0.22)	0.7
	s_6	(0.25, 0.70)	0.4
b_{S4}	s_7	(0.58, 0.50)	0.4
	s_8	(0.68, 0.42)	0.2

(b) collection S

BA Example:
$k = 1, \epsilon = 0.1, \gamma = SUM$

Round 1:
Build A_{R1} on block b_{R1}
Join A_{R1} with $\emptyset$
$C = \{\}$

Block-based Algorithm (BA)

- Example

	id	*loc*	*score*
b_{R1}	r_1	(0.20, 0.78)	1.0
	r_2	(0.30, 0.64)	0.8
b_{R2}	r_3	(0.20, 0.45)	0.8
	r_4	(0.40, 0.90)	0.6
b_{R3}	r_5	(0.63, 0.12)	0.6
	r_6	(0.91, 0.63)	0.4
b_{R4}	r_7	(0.79, 0.20)	0.3
	r_8	(0.76, 0.42)	0.1

(a) collection R

	id	*loc*	*score*
b_{S1}	s_1	(0.69, 0.85)	0.9
	s_2	(0.81, 0.71)	0.9
b_{S2}	s_3	(0.24, 0.38)	0.8
	s_4	(0.15, 0.52)	0.7
b_{S3}	s_5	(0.40, 0.22)	0.7
	s_6	(0.25, 0.70)	0.4
b_{S4}	s_7	(0.58, 0.50)	0.4
	s_8	(0.68, 0.42)	0.2

(b) collection S

BA Example:
$k = 1, \epsilon = 0.1, \gamma = SUM$

Round 2:
Build A_{S1} on block b_{S1}
Join A_{S1} with A_{R1}
$C = \{\}$

Block-based Algorithm (BA)

- Example

	id	*loc*	*score*
b_{R1}	r_1	(0.20, 0.78)	1.0
	r_2	(0.30, 0.64)	0.8
b_{R2}	r_3	(0.20, 0.45)	0.8
	r_4	(0.40, 0.90)	0.6
b_{R3}	r_5	(0.63, 0.12)	0.6
	r_6	(0.91, 0.63)	0.4
b_{R4}	r_7	(0.79, 0.20)	0.3
	r_8	(0.76, 0.42)	0.1

(a) collection R

	id	*loc*	*score*
b_{S1}	s_1	(0.69, 0.85)	0.9
	s_2	(0.81, 0.71)	0.9
b_{S2}	s_3	(0.24, 0.38)	0.8
	s_4	(0.15, 0.52)	0.7
b_{S3}	s_5	(0.40, 0.22)	0.7
	s_6	(0.25, 0.70)	0.4
b_{S4}	s_7	(0.58, 0.50)	0.4
	s_8	(0.68, 0.42)	0.2

(b) collection S

BA Example:
$k = 1, \epsilon = 0.1, \gamma = SUM$

Round 3:
Build A_{S2} on block b_{S2}
Join A_{S2} with A_{R1}
$C = \{\}$

Block-based Algorithm (BA)

- Example

	id	*loc*	*score*
b_{R1}	r_1	(0.20, 0.78)	1.0
	r_2	(0.30, 0.64)	0.8
b_{R2}	r_3	(0.20, 0.45)	0.8
	r_4	(0.40, 0.90)	0.6
b_{R3}	r_5	(0.63, 0.12)	0.6
	r_6	(0.91, 0.63)	0.4
b_{R4}	r_7	(0.79, 0.20)	0.3
	r_8	(0.76, 0.42)	0.1

(a) collection R

	id	*loc*	*score*
b_{S1}	s_1	(0.69, 0.85)	0.9
	s_2	(0.81, 0.71)	0.9
b_{S2}	s_3	(0.24, 0.38)	0.8
	s_4	(0.15, 0.52)	0.7
b_{S3}	s_5	(0.40, 0.22)	0.7
	s_6	(0.25, 0.70)	0.4
b_{S4}	s_7	(0.58, 0.50)	0.4
	s_8	(0.68, 0.42)	0.2

(b) collection S

BA Example:
$k = 1, \epsilon = 0.1, \gamma = SUM$

Round 4:
Build A_{R2} on block b_{R2}
Join A_{R2} with A_{S1} and A_{S2}
$C = \{(r_3, s_3)\}$

Block-based Algorithm (BA)

- Example

	id	*loc*	*score*
b_{R1}	r_1	(0.20, 0.78)	1.0
	r_2	(0.30, 0.64)	0.8
b_{R2}	r_3	(0.20, 0.45)	0.8
	r_4	(0.40, 0.90)	0.6
b_{R3}	r_5	(0.63, 0.12)	0.6
	r_6	(0.91, 0.63)	0.4
b_{R4}	r_7	(0.79, 0.20)	0.3
	r_8	(0.76, 0.42)	0.1

(a) collection R

	id	*loc*	*score*
b_{S1}	s_1	(0.69, 0.85)	0.9
	s_2	(0.81, 0.71)	0.9
b_{S2}	s_3	(0.24, 0.38)	0.8
	s_4	(0.15, 0.52)	0.7
b_{S3}	s_5	(0.40, 0.22)	0.7
	s_6	(0.25, 0.70)	0.4
b_{S4}	s_7	(0.58, 0.50)	0.4
	s_8	(0.68, 0.42)	0.2

(b) collection S

BA Example:
$k = 1, \epsilon = 0.1, \gamma = SUM$

Round 5:
Build A_{S3} on block b_{S3}
Join A_{S3} with A_{R1}
$C = \{(r_3, s_3) = 1.6\}$

Note:
A_{S3} won't join with A_{R2}
$\gamma(b^u_{R2}, b^u_{S3}) = 1.5 < 1.6$

$l_R = 0.6, l_S = 0.4$
$T = 1.5 < \gamma(r_3, s_3)$
return C

Block-based Algorithm (BA)

- Access on block level
  - Avoid indexing whole datasets
- Bulk-load and join blocks
  - Avoid repeated update and range query
- Score bound for each block
  - Block-based pruning

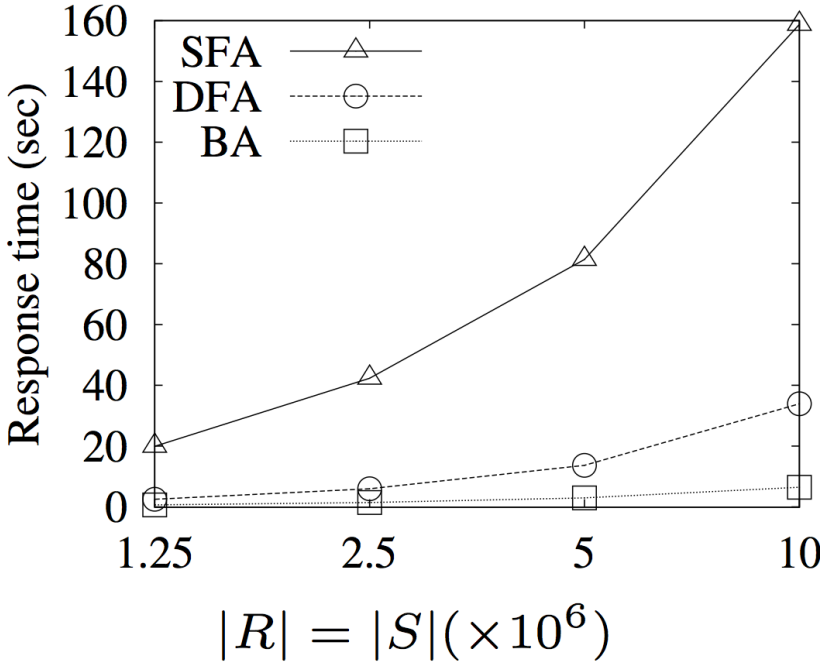
Outline

- Related Work
- Algorithms
- Experiments
- Conclusions

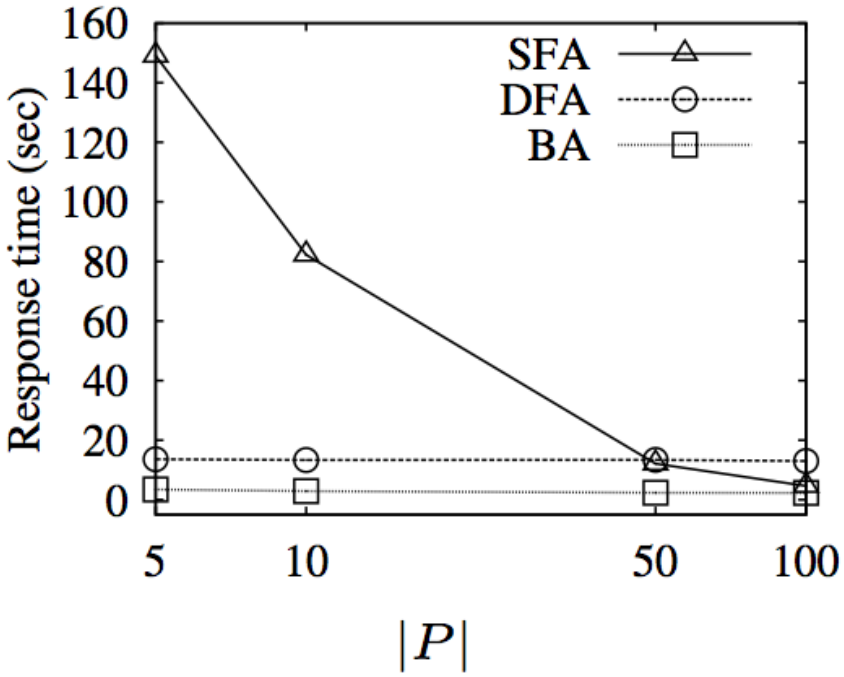
Experimental Evaluation

- Datasets
  - FLICKR, 1.68M locations in London
  - ISLES, 20M POIs from OpenStreetMap
  - Tag scores according to $|P|$ seed points (simulating POIs) [Yiu 2011]
    - $o.score = 1 - \min_{i=1}^{|P|} dist(o, P_i)$
- Assumptions
  - $\gamma = SUM$
  - Data in memory

Data

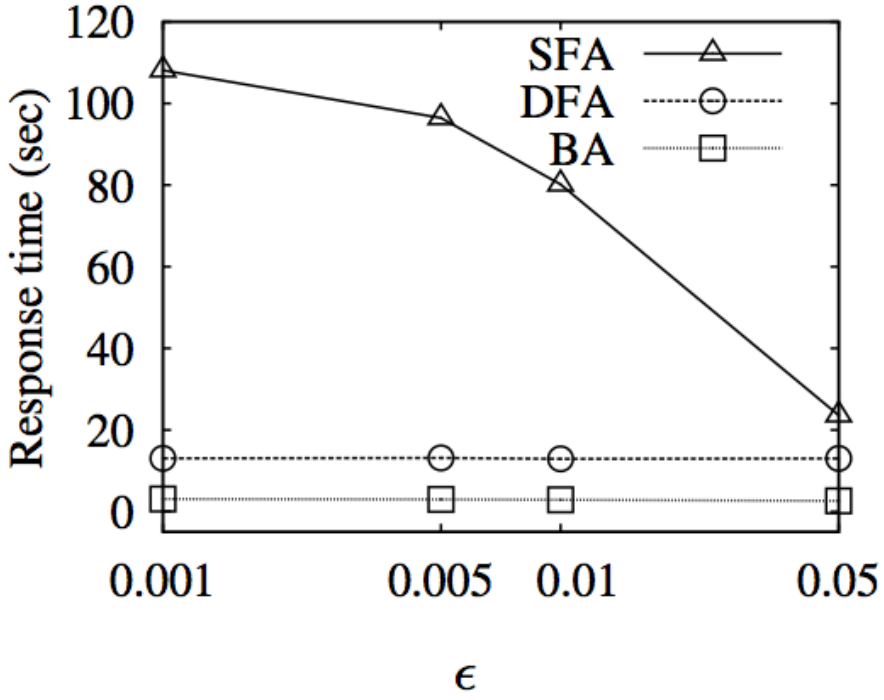


Dataset cardinality

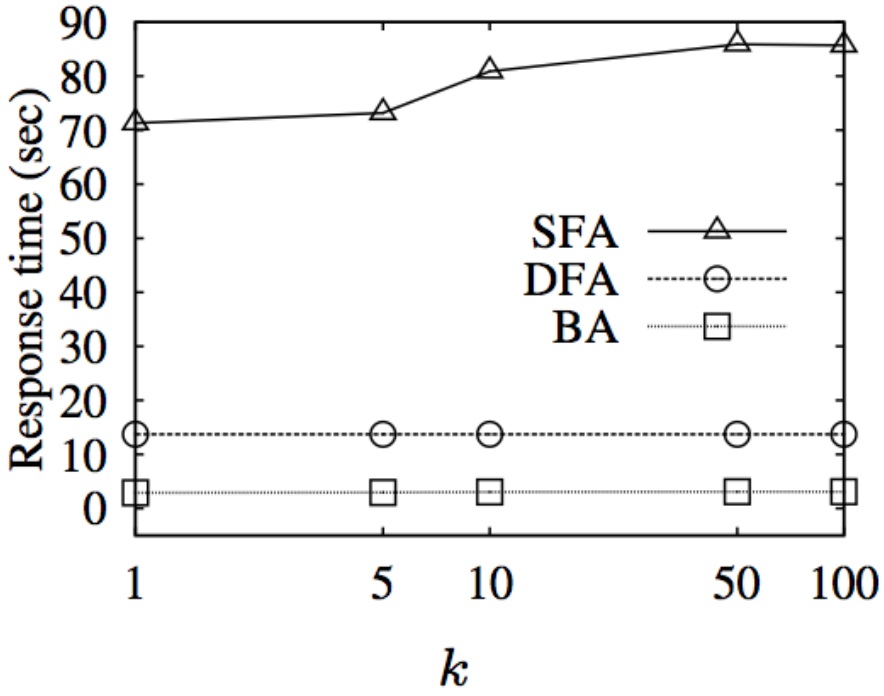


Number of seeds

Parameters



Distance threshold



Number of expected results

Outline

- Related Work
- Algorithms
- Experiments
- Conclusions

To sum up

- Conclusions
  - We proposed and studied the top-*k* spatial distance join query
  - We proposed three *k*-SDJ solutions, i.e., SFA, DFA and BA
  - According to our experiments, BA performs best in practice
- Future work
  - BA as general framework for complex top-*k* join queries (e.g., top-*k* string join)
  - Theoretical analysis on optimal block size selection
  - Combined distance/score ranking

Thank you

References

- [Arge 1998] Arge, L., Procopiuc, O., Ramaswamy, S., Suel, T., Vitter, J.S.: Scalable sweeping based spatial join. In: VLDB. (1998) 570-581
- [Brinkhoff 1993] Brinkhoff, T., Kriegel, H.P., Seeger, B.: Efficient processing of spatial joins using R-trees. In: SIGMOD Conference. (1993)
- [Ilyas 2003] Ilyas, I.F., Aref, W.G., Elmagarmid, A.K.: Supporting top-k join queries in relational databases. In: VLDB. (2003) 754-765
- [Ilyas 2008] Ilyas, I.F., Beskales, G., Soliman, M.A.: A survey of top-k query processing techniques in relational database systems. ACM Comput. Surv. 40(4) (2008)
- [Yiu 2011] Yiu, M.L., Lu, H., Mamoulis, N., Vaitis, M.: Ranking spatial data by quality preferences. TKDE. 23(3) (2011) 433-446