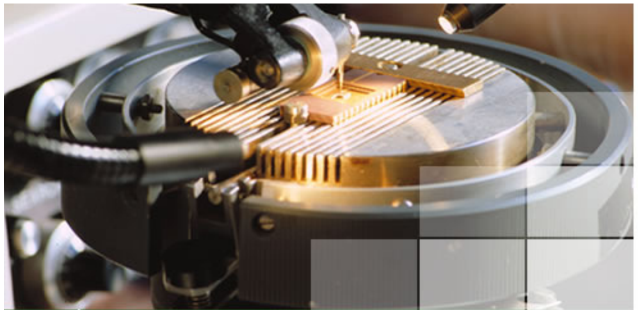


POLITECNICO DI MILANO

Dipartimento di
Elettronica e Informazione

Rank Aggregation and Rank join in Search Computing

Davide Martinenghi

December 1, 2011

Outline

- The rank aggregation problem
- Methods for rank aggregation
  - Median-based
  - With objective function
- Rank aggregation in Search Computing
  - Similarities and differences
  - Challenges
- Rank join
  - Similarities and differences
  - The Pull-Bound Rank Join template
- Further extensions
  - Proximity rank join
  - Uncertainty in rank join

Rank aggregation

- Rank aggregation is the problem of combining several ranked lists of objects in a robust way to produce a single consensus ranking of the objects
- Main applications of rank aggregation:
  - Combination of user preferences expressed according to various criteria
    - Example: ranking restaurants by combining criteria about culinary preference, driving distance, stars, …
  - Nearest neighbor problem (e.g., similarity search)
    - Given a database *D* of *n* points in some metric space, and a query *q* in the same space, find the point (or the *k* points) in *D* closest to *q*
  - Search computing
    - …

Main approaches to rank aggregation

- Axiomatic approach
  - Desiderata of aggregation function formulated as “axioms”
  - By the classical result of Arrow, a small set of natural requirements cannot be simultaneously achieved by any nontrivial aggregation function

- Metric approach
  - Finding a new ranking R whose total distance to the initial rankings $R_1, \ldots, R_n$ is minimized
  - For several metrics, NP-hard to solve exactly
    - E.g., the **Kendall tau distance** $K(R_1, R_2)$, defined as the number of exchanges in a bubble sort to convert R_1 to R_n
  - May admit efficient approximations

Rank aggregation in data-centric contexts

- Data attributes may be non-numeric
  - Even if they are numeric, differences within certain ranges may not matter to the user
  - Rankings may have ties between elements (partial rankings)
- Traditionally, two ways of accessing data:
  - Random access: given an element, retrieve its score (position in the ranked list or other associated value)
  - Sequential access: access, one by one, the next element (together with its score) in a ranked list, starting from the top
- Main interest in the top k elements of the aggregation
  - Need for algorithms that quickly obtain the top results
  - … without having to read each ranking in its entirety
- Several algorithms developed in the literature to minimize the accesses when determining the top k elements
  - Main works by Fagin et al.

Combining opaque rankings

- Techniques using only the position of the elements in the ranking (no other associated score)
- We review MedRank, proposed by Fagin et al.
  - An algorithm for rank aggregation based on the notion of median

Input: *m* rankings of *n* elements

Output: the top *k* elements in the aggregated ranking

1. **Use sequential accesses in each ranking, one element at a time, until there are k elements that occur in more than m/2 rankings**
2. **These are the top k elements**

- MedRank is instance-optimal
  - Among the algorithms that access the rankings in sequential order, this algorithm is the best possible algorithm (to within a constant factor) on every input instance

MedRank example: hotels in Paris

Hotels by price	Hotels by rating
Ibis	Crillon
Etap	Novotel
Novotel	Sheraton
Mercure	Hilton
Hilton	Ibis
Sheraton	Ritz
Crillon	Lutetia
…	…

Top 3 hotels
Novotel
Hilton
Ibis

- Strategy:
  - Make one sequential access at a time in each ranking
  - Look for hotels that appear in both rankings

NB: price and rating are opaque, only the position matters

Combining ranking with an *objective function*

- Several studies consider rankings where the objects, besides the position, also include a score given by a truth value in the [0, 1] interval
  - Truth gives the degree of matching between the object and the given criterion
- Example query: “names of albums by the Beatles whose cover color is red”
  - Being an album by the Beatles has a crisp truth value (0 or 1)
  - “Redness” of the cover has a fuzzy truth value in [0, 1]
- Fagin has considered queries combining objects with such truth values by Boolean operators
  - Monotonic operators: *min* for conjunction, *max* for disjunction

Fagin's algorithm for monotone queries

Input: a monotone query combining rankings $R_1, \ldots, R_n$

Output: the top *k* <object, score> pairs

1. **Extract the same number of objects by sequential accesses in each ranking until there are at least *k* objects that match the query**
2. **For each extracted object, compute its overall score by making random accesses wherever needed**
3. **Among these, output the *k* objects with the best overall score**

- Complexity is sub-linear in the number *N* of objects
  - Proportional to the square root of *N* when combining two rankings

Example cont'd: hotels in Paris

Hotels	Cheapness	Hotels	Rating
Ibis	.92	Crillon	.9
Etap	.91	Novotel	.9
Novotel	.85	Sheraton	.8
Mercure	.85	Hilton	.7
Hilton	.825	Ibis	.7
Sheraton	.8	Ritz	.7
Crillon	.75	Lutetia	.6
…		…	

Top 3	Score
Novotel	.85
Sheraton	.8
Crillon	.75

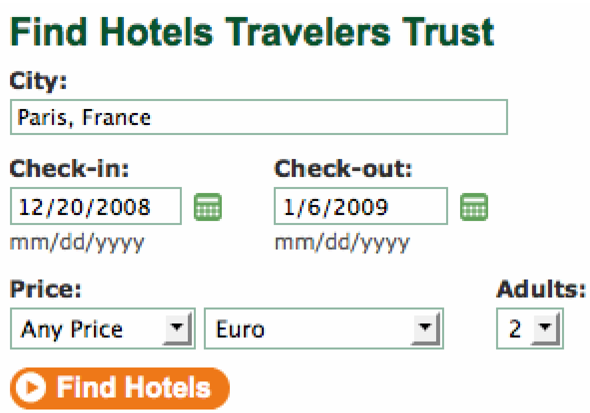
- Query: hotels with best price and rating
- Strategy:
  - Make one sequential access at a time in each ranking
  - Look for hotels that appear in both rankings

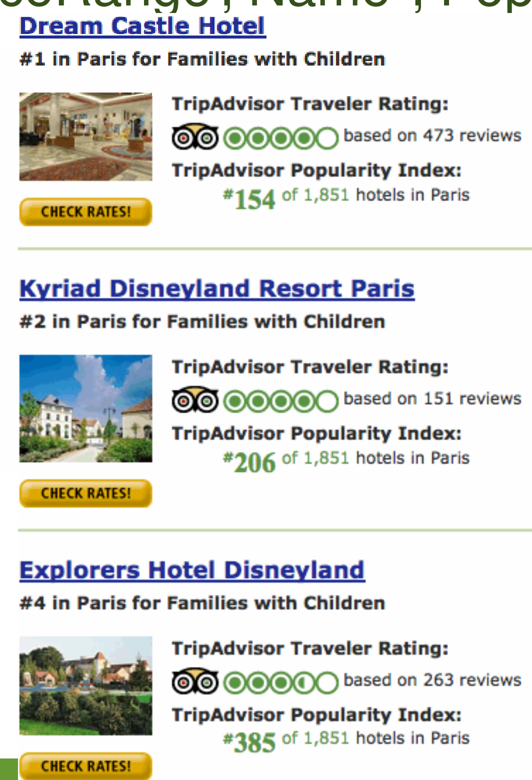
NB: price and rating are used to compute the overall score

Characterizing services in search computing

- Service model similar to relational model, but
  - Services expose a limited number of interfaces with input and output fields (access patterns)
  - Results of a service call are typically ranked
  - Example:

    tripAdvisor(City^{i}, InDate^{i}, OutDate^{i}, Persons^{i}, PriceRange^{i}, Name^{o}, Popularity$^{o,\text{ranked}}$)





Characterizing services in search computing

- Service model similar to relational model, but
  - Services expose a limited number of interfaces with input and output fields (access patterns)
  - Results of a service call are typically ranked
  - Example:
    tripAdvisor(City^{i}, InDate^{i}, OutDate^{i}, Persons^{i}, PriceRange^{i}, Name^{o}, Popularity^{o,ranked})
- Service invocations (accesses) may also be associated with costs, such as the average response time
- Services are further classified as:
  - Proliferative (in average, more than one result) or selective
  - Chunked (results come in pages) or bulk (all in one shot)
- Can we reuse rank aggregation techniques to combine services' rankings in query results?

Challenges of rank aggregation in search computing

- Availability of accesses depends on access patterns
  - Example: Sequential access for hotels by rating in MedRank

    tripAdvisor(City^{i}, InDate^{i}, OutDate^{i}, Persons^{i}, PriceRange^{i}, Name^{o}, Popularity^{o,ranked})
  - NB: the size of a page of results is not necessarily 1
- Rankings are permutations of all the objects of a domain
  - Services' results do not necessarily contain all objects
- Scores are not necessarily in the [0,1] interval
  - In the example we have euros and ratings…
- Meaningful objective functions different from *min*
  - E.g., best combination of price of hotel + flight within a given threshold
- What are the objects?
  - In the example, hotel names are the identifiers
  - In general, there will join conditions to be satisfied

- Given
  - A natural join $R_1 \bowtie R_2$
  - A score aggregation function ***S***
  - A positive integer $k < |R_1 \bowtie R_2|$
  - Some additional assumptions (to be described).
- Compute
  - k join results with highest scores

(part of the material on rank join courtesy of Neoklis Polyzotis)

Relational Top-K Queries


```
SELECT h.neighborhood, h.hid, r.rid
FROM HotelsNY h, RestaurantsNY r
WHERE h.neighborhood = r.neighborhood
RANK BY 0.4/h.price + 0.4*r.rating + 0.2*r.hasMusic
LIMIT 5
```

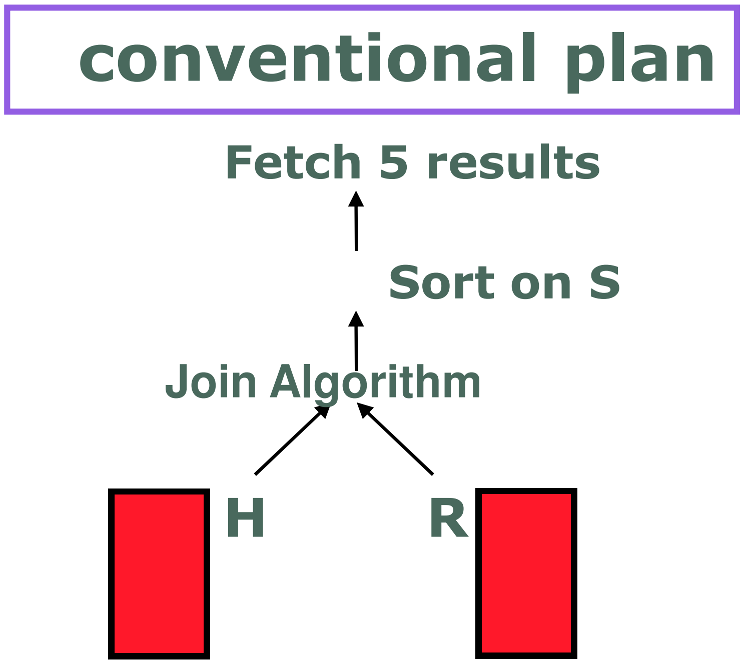
Full Join Results

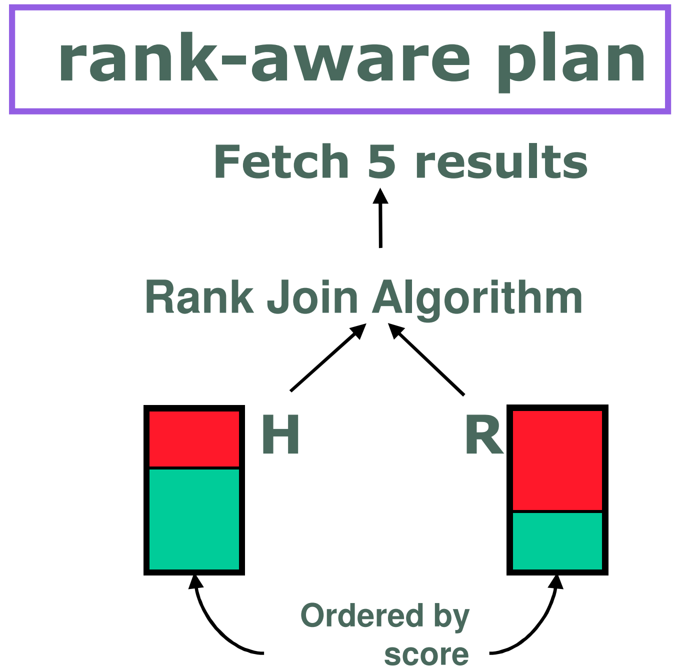
Neighborhood	Hid	Rid
West Village	H89	R585
Midtown East	H248	R197
Chelsea	H427	R572
Midtown East	H248	R346
Midtown East	H597	R197
Hell's Kitchen	H662	R223
Midtown West	H141	R276
Upper East Side	H978	R137
Harlem	H355	R49
Tribeca	H381	R938
. . .	. . .	. . .

Rank Join Results

Neighborhood	Hid	Rid
East Village	H346	R738
Gramercy	H872	R822
Midtown West	H141	R276
Hell's Kitchen	H662	R498
Upper West Side	H51	R394

Ranking Query Evaluation





Rank Join Algorithms

- First desideratum: early termination
  – No algorithm can be optimal
  – But an algorithm can be instance optimal
- Second desideratum: computational efficiency
- State of the art: HRJN* (Hash Rank JoiN)
  – Computationally efficient
  – **Instance optimal** in some cases

Scoring Function


```
SELECT h.neighborhood, h.hid, r.rid
FROM HotelsNY h, RestaurantsNY r
WHERE h.neighborhood = r.neighborhood
RANK BY 0.4/h.price + 0.4*r.rating + 0.2*r.hasMusic
LIMIT 5
```

- A base score vector for each tuple

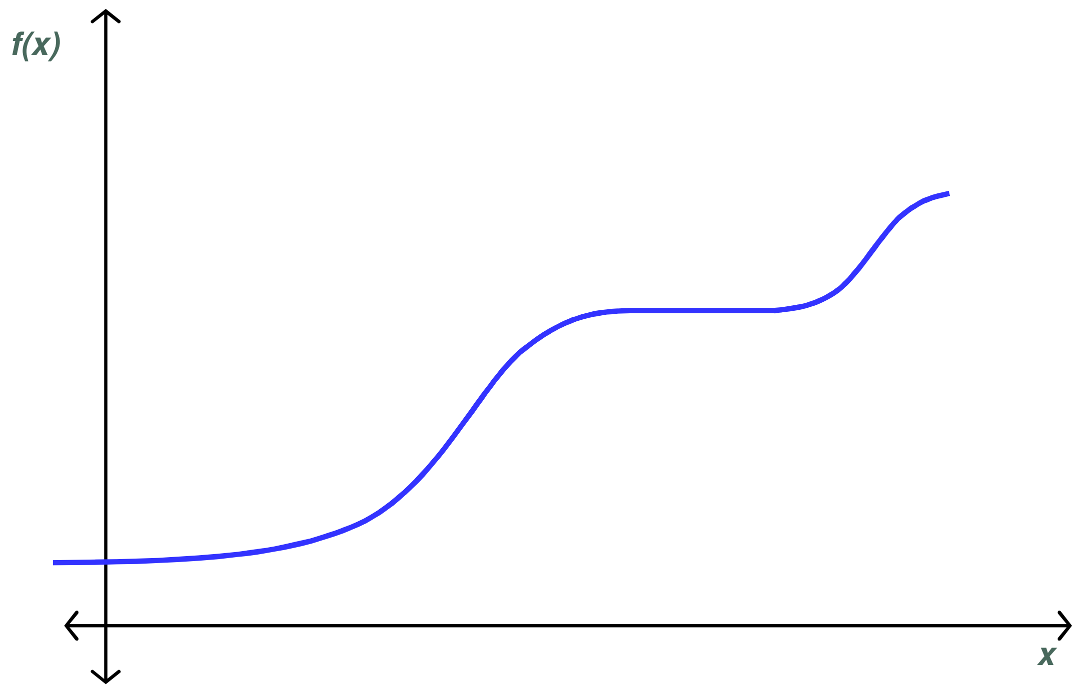
  Hotels: **b**(h) = [1/h.price] $\in [0,1]$

  Restaurants: **b**(r) = [r.rating, r.hasMusic] $\in [0,1]^2$

- Combined with a scoring function S
  - S(h1, r1, r2) = 0.4*h1 + 0.4*r1 + 0.2*r2

Monotonic Functions

- A function f(x1,...,xn) is monotonic if
  - $\forall i\ (x_i \le y_i) \Rightarrow f(x_1,\dots,x_n) \le f(y_1,\dots,y_n)$



The Scoring Function

- We require that ***S*** is monotonic
- Monotonicity enables bounds on score
  - Recall example, we had a weighted average

$$\boldsymbol{S}(h_1, r_1, r_2) = 0.4*h_1 + 0.4*r_1 + 0.2*r_2$$

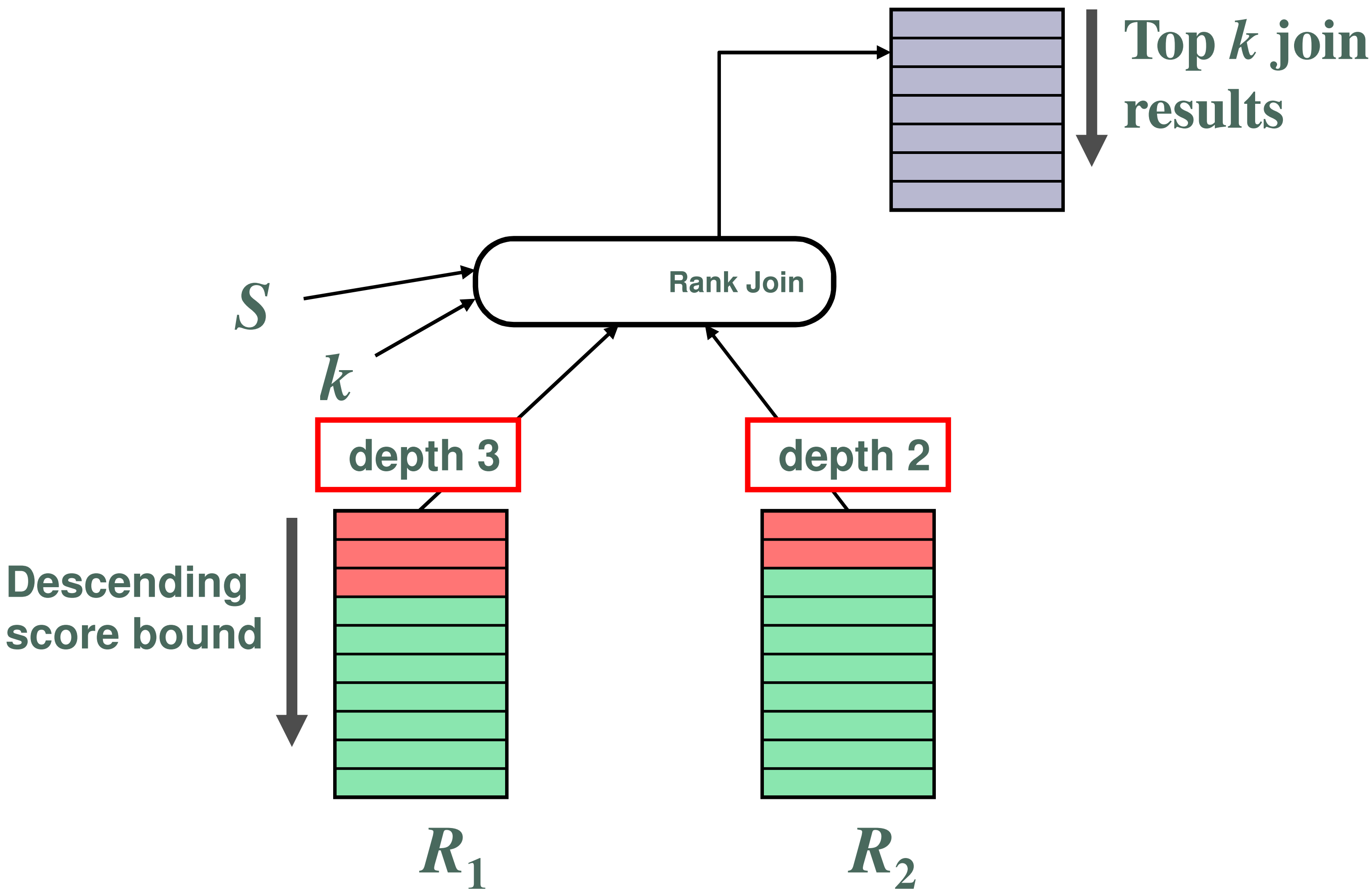
  - The *score bound* of a hotel η is defined as

$$\begin{aligned}\overline{\boldsymbol{S}}(\eta) &= \boldsymbol{S}(1/\eta.\text{price}, 1, 1)\\ &= 0.4/\eta.\text{price} + 0.4*1 + 0.2*1\end{aligned}$$

  - By monotonicity, for any restaurant tuple ρ,

$$\boldsymbol{S}(\eta \bowtie \rho) \leq \overline{\boldsymbol{S}}(\eta)$$

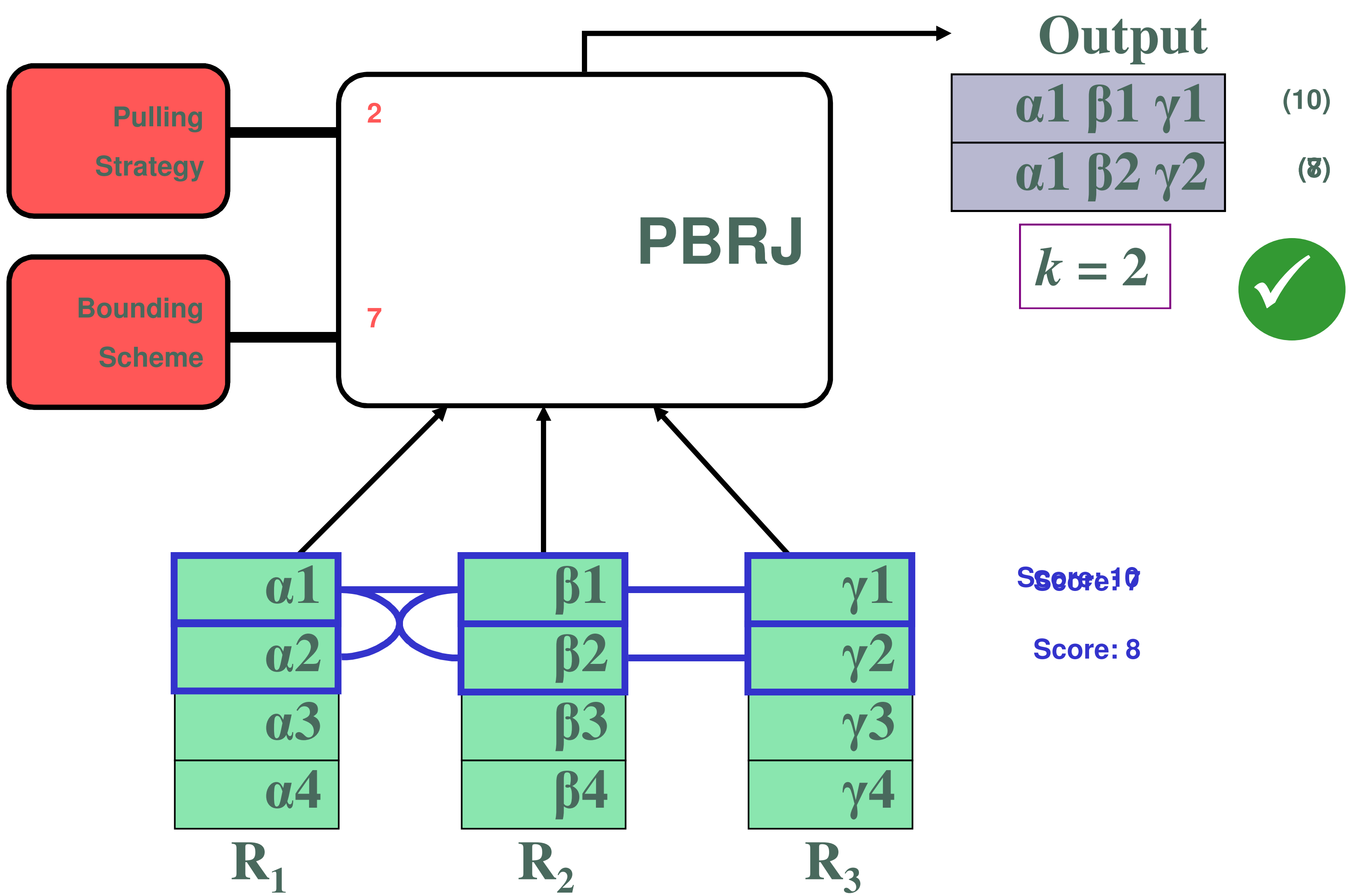
The Rank Join Problem



Performance Metrics

- We use total depth (aka sumDepths) as our primary cost metric.
  - cost(A, I) = $\sum_{i=1}^{n}$ depth on Ri
- We have formal optimality concepts
  - Fix sets of algorithms B and problem instances J
  - H ∈ B is optimal if:
    - cost(H, I) ≤ cost(A, I) for all A ∈ B and I ∈ J
  - H ∈ B is instance-optimal if:
    - cost(H, I) ≤ c1 · cost(A, I) + c2 for all A ∈ B and I ∈ J and some constants c1, c2

Pull/Bound Rank Join



The HRJN* algorithm

- Bounding scheme: corner bound (see below)
- Pulling strategy: choose input with highest bound

Att	b1	$\overline{S}$
x	1.0	3.0
y	0.7	2.9
z	0.5	2.5

R_1

Att	b2	$\overline{S}$
w	1.0	3.0
x	0.8	2.7

R_2

Att	b3	$\overline{S}$
z	1.0	3.0
x	0.9	2.9

R_3

Record last score bound on each input and take the max

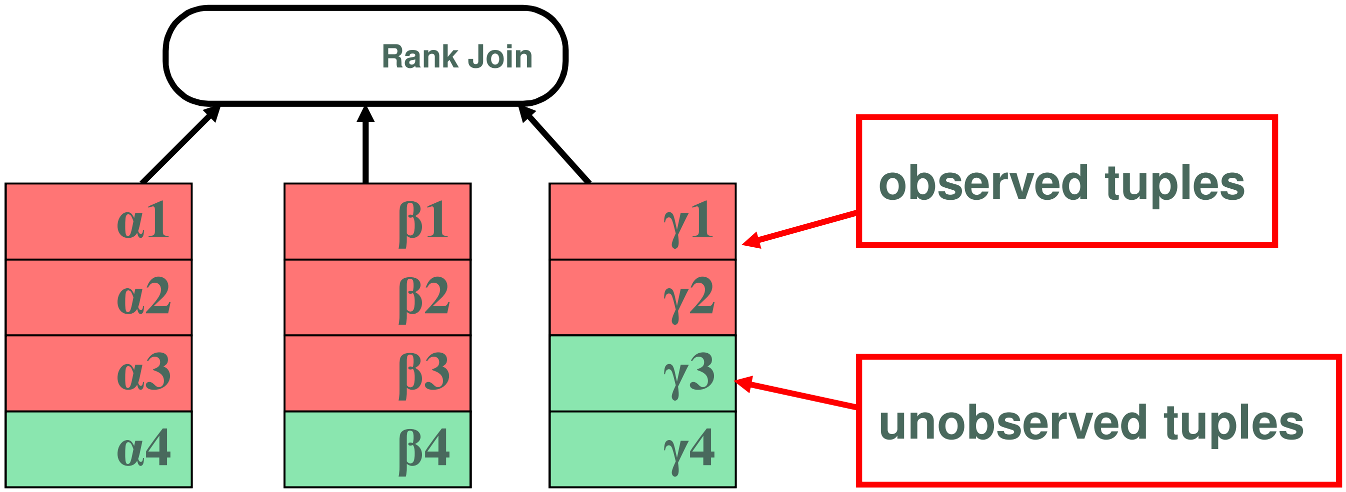
$$\begin{aligned} t_1 &= 2.5 \\ t_2 &= 2.7 \\ t_3 &= 2.9 \\ t &= \max(t_1, t_2, t_3) \\ &= 2.9 \end{aligned}$$

Analysis of the Corner Bound

- Let Fc be the set of PBRJ algorithms using the corner bound and any pulling strategy
- Positive results
  - HRJN* is optimal within Fc
- Negative result
  - No member of Fc is instance optimal within all correct deterministic algorithms
  - Problem: The corner bound is loose (not tight)

Tight Bounding

- Definition: A bounding scheme is tight if it always returns a score that can be observed from unknown join results



Theorem: PBRJ is instance-optimal if it uses

- **a tight bounding scheme, and**
- **the round-robin pulling strategy**

- Downside: tight bounds are slower to compute

Proximity Rank Join: example

- A smartphone user wants to organize the evening by finding:
  - a restaurant, a movie theater and a hotel that are
    - nearby
    - close to each other
    - recommended in terms of price, user rating, and number of stars

Proximity rank join problems

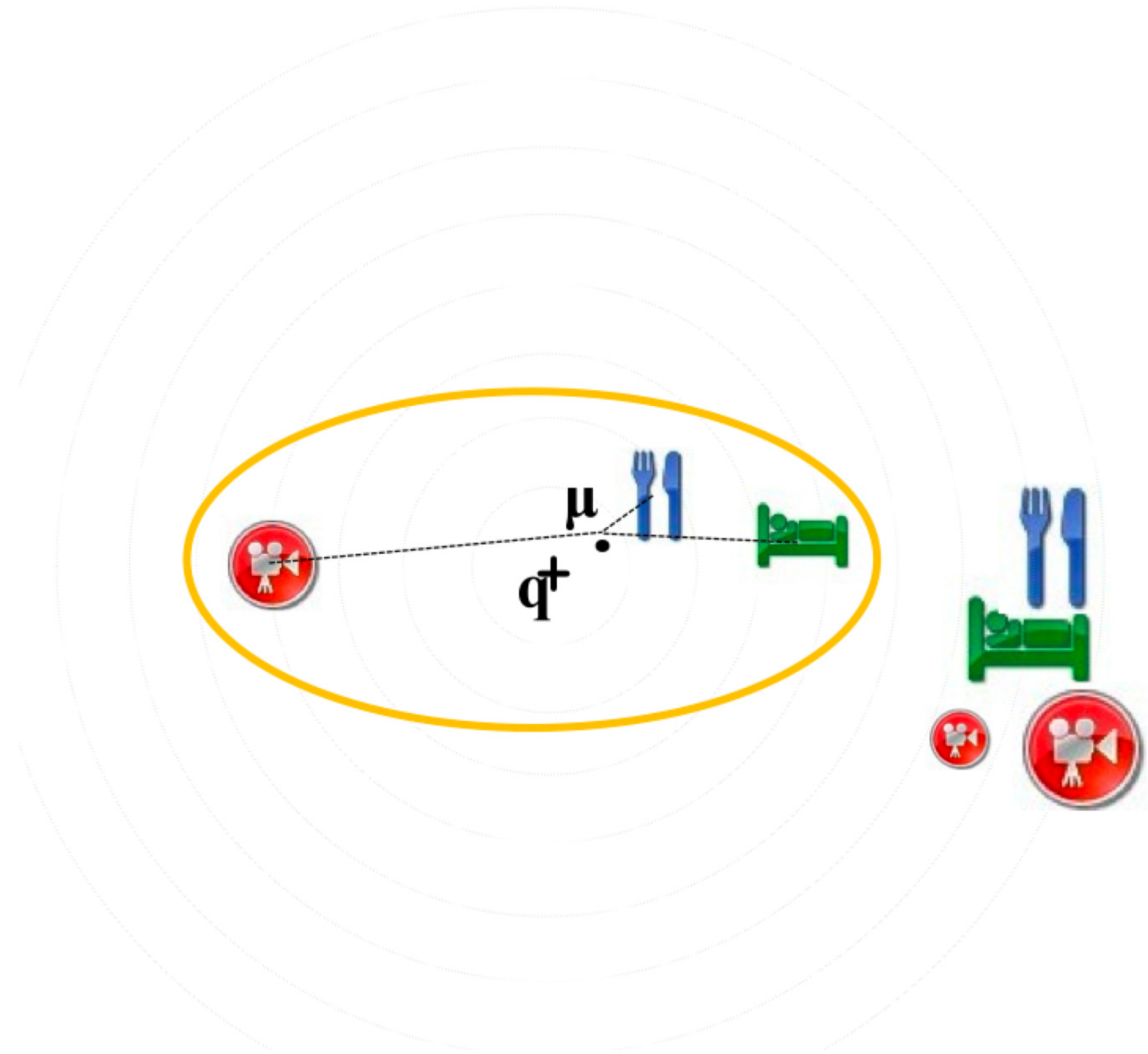
- Look for combinations of objects coming from different services
- Each object is equipped with
  - A score
  - A real valued feature vector

Hotel	Category	Location
Villa D'Este	5	[45.62 N, 9.32 E]
Metropole Suisse	4	[45.65 N, 9.33 E]
Palace Hotel	4	[45.64 N, 9.31 E]

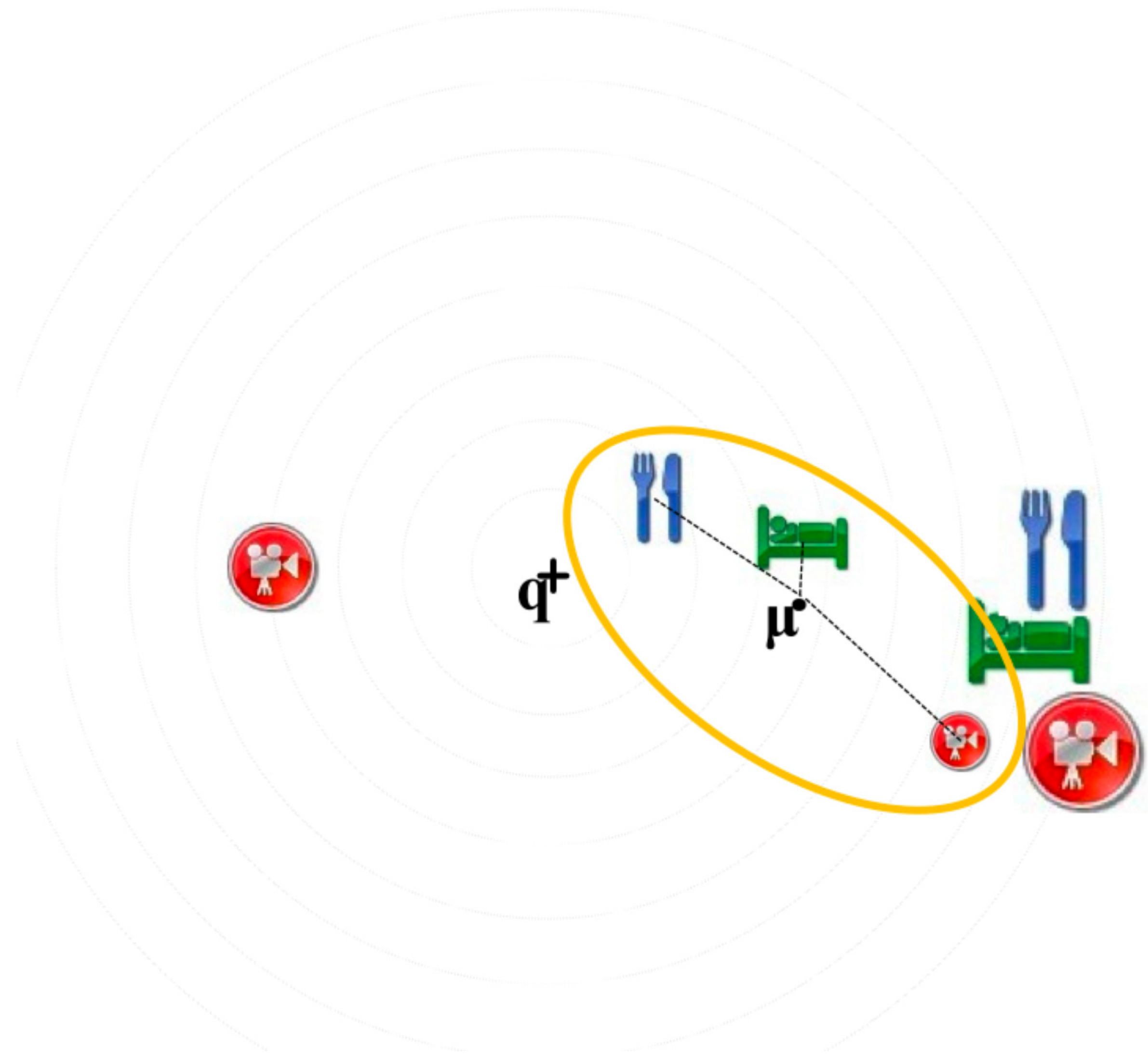
Proximity rank join problems

- The aggregation function assigns a score to a combination based on
  - The individual scores
  - The proximity to the query vector
  - Their mutual proximity

Proximity Rank Join



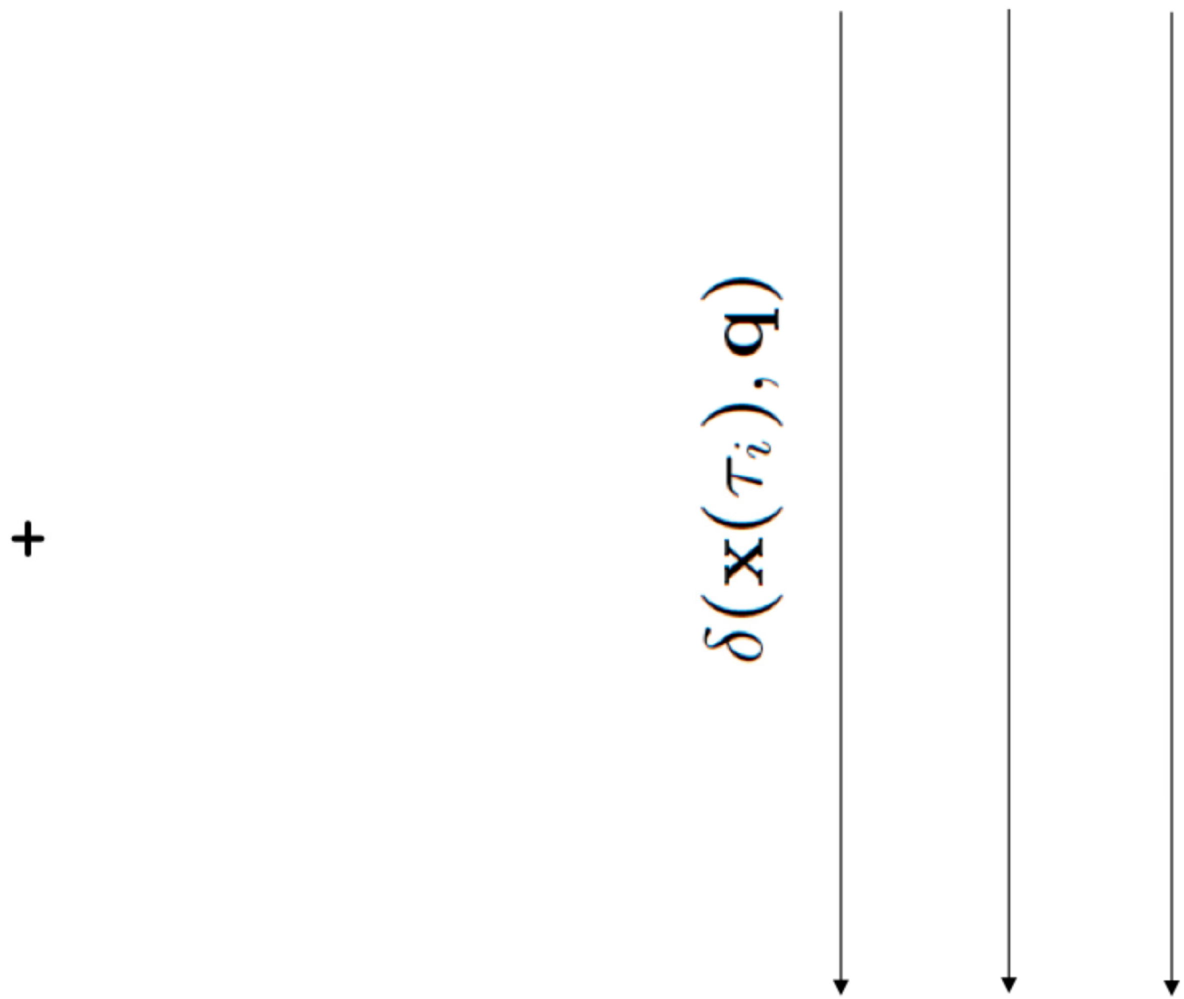
Proximity Rank Join



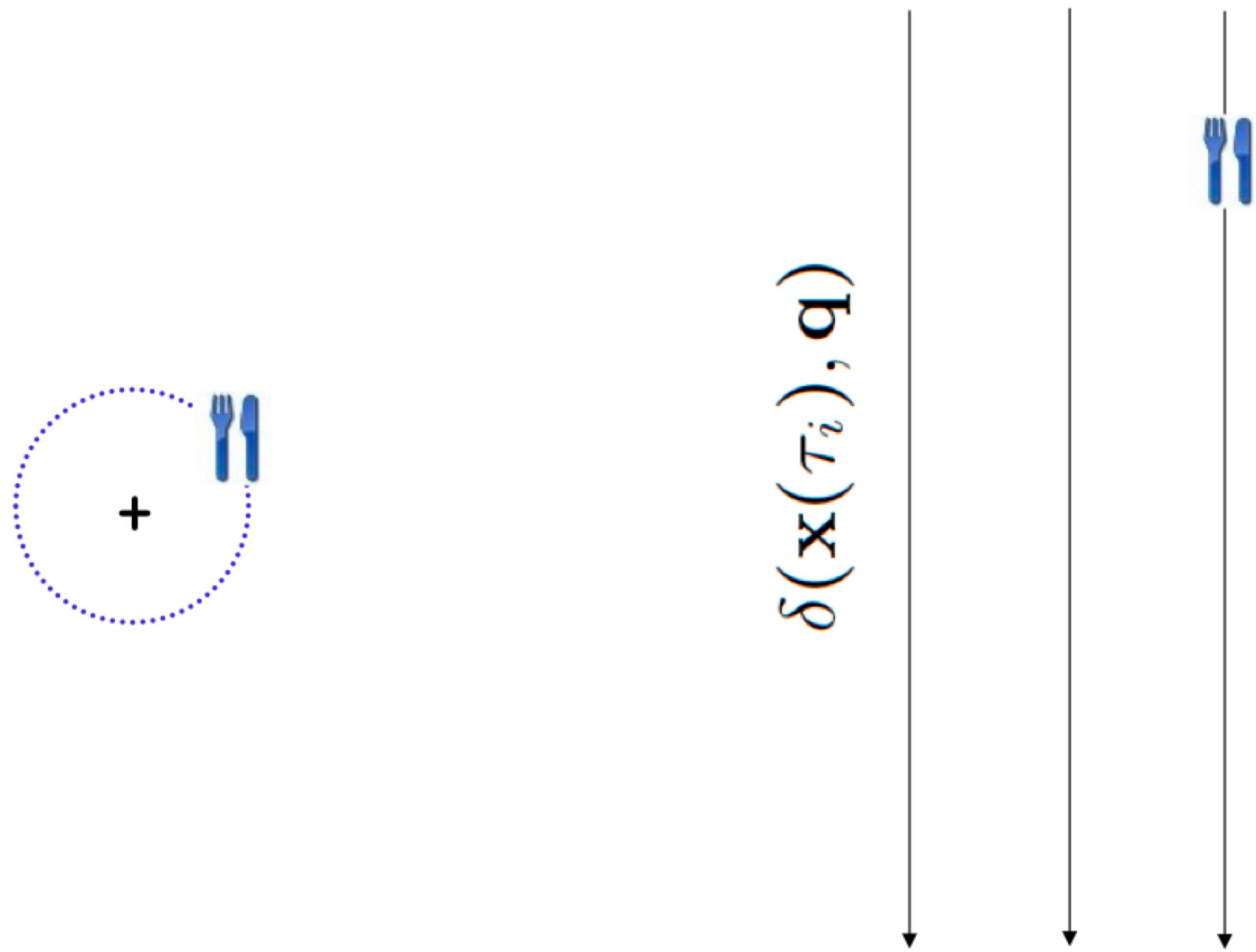
Proximity rank join problems

- Objects can be retrieved sorted by
  - Distance from the query
  - Score (not shown in this presentation)

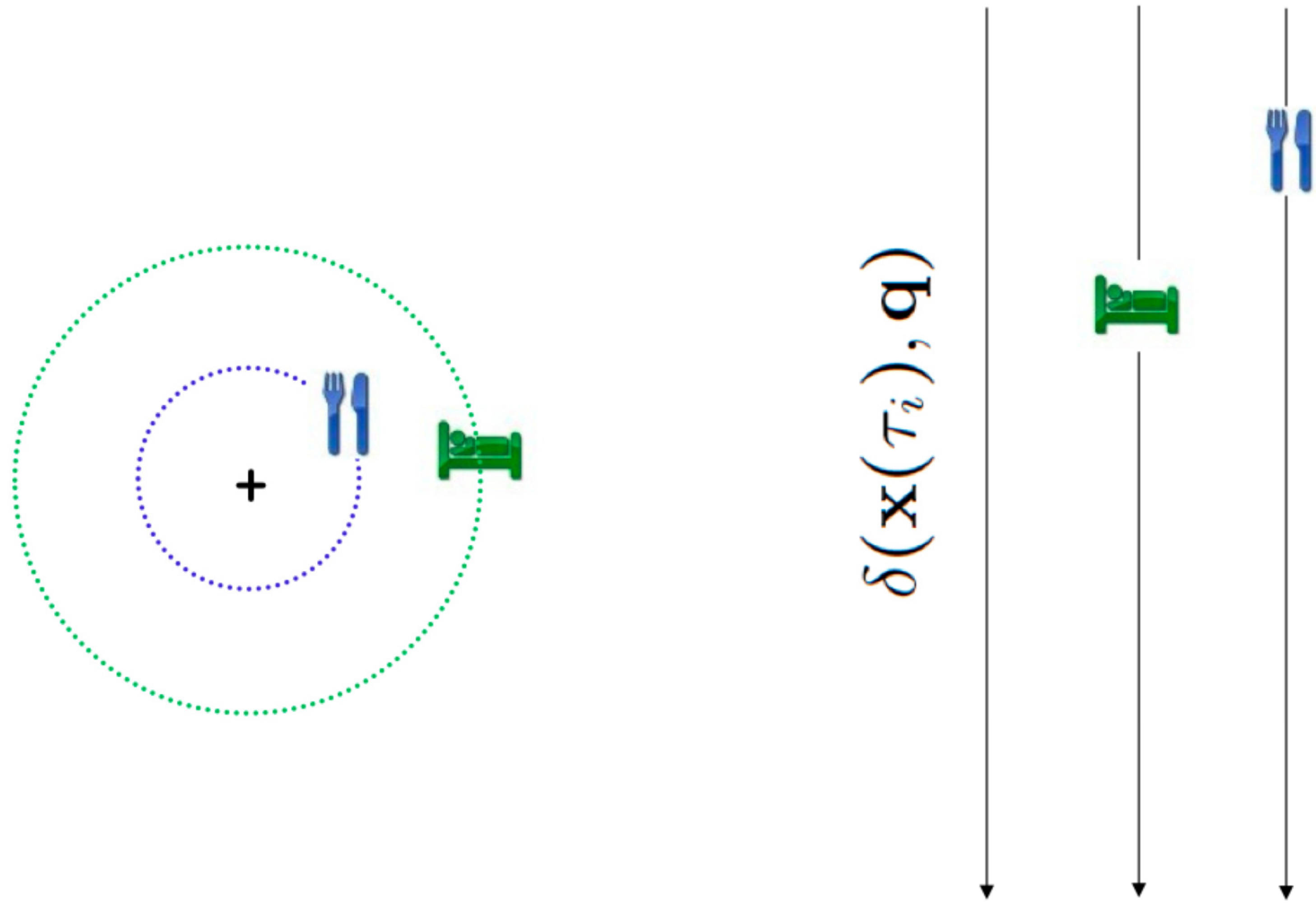
Distance-based access



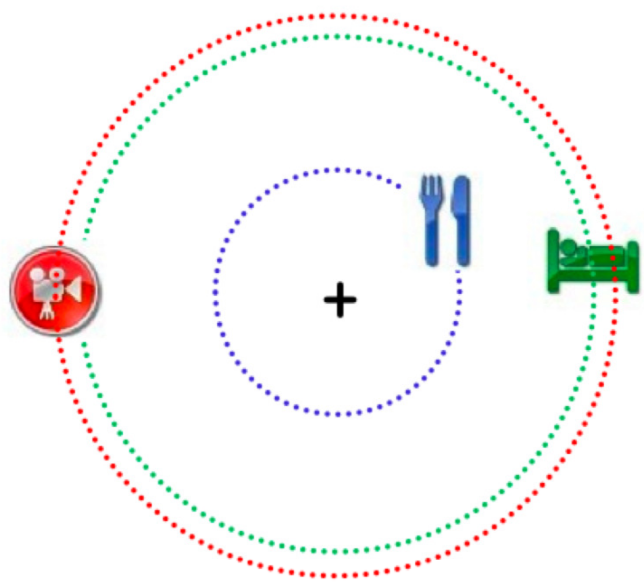
Distance-based access

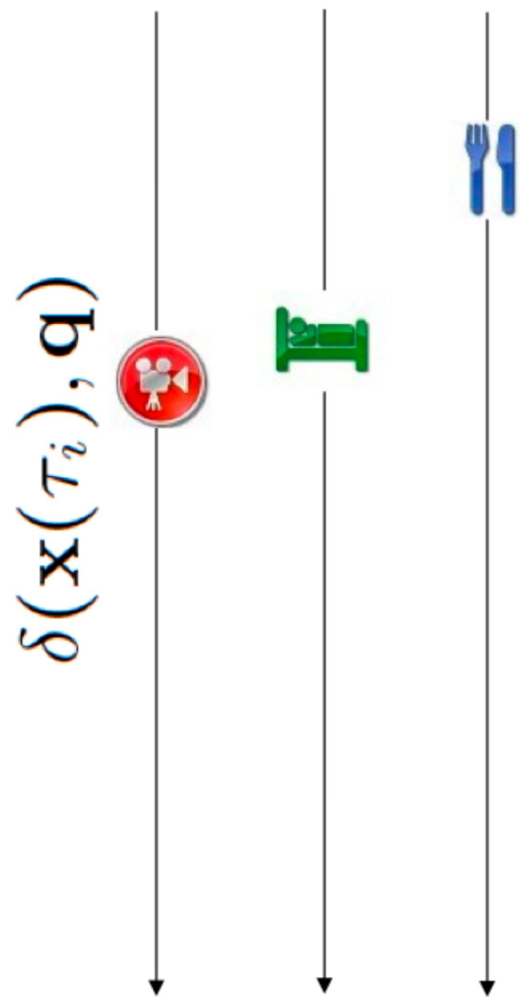


Distance-based access

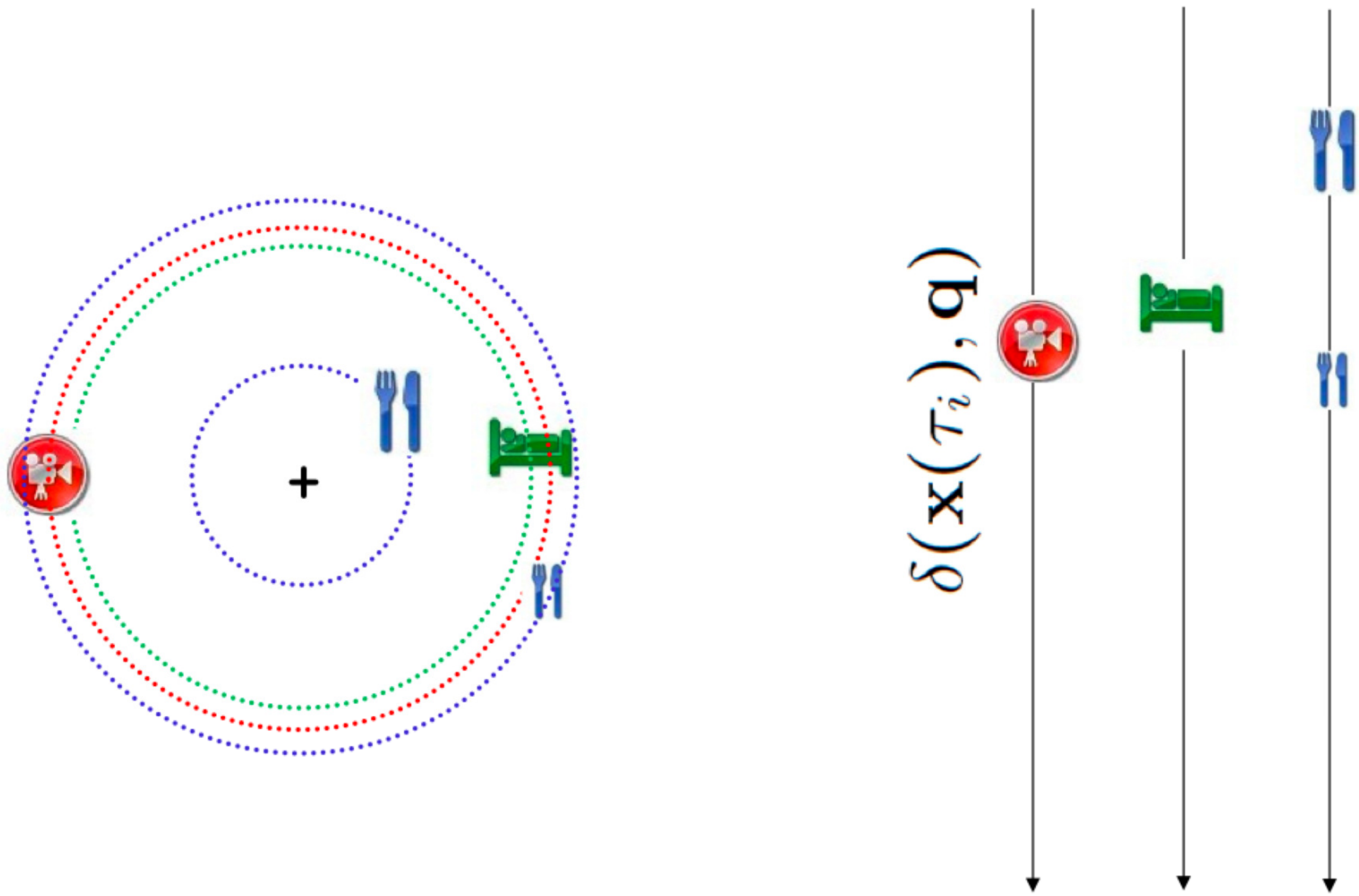


Distance-based access

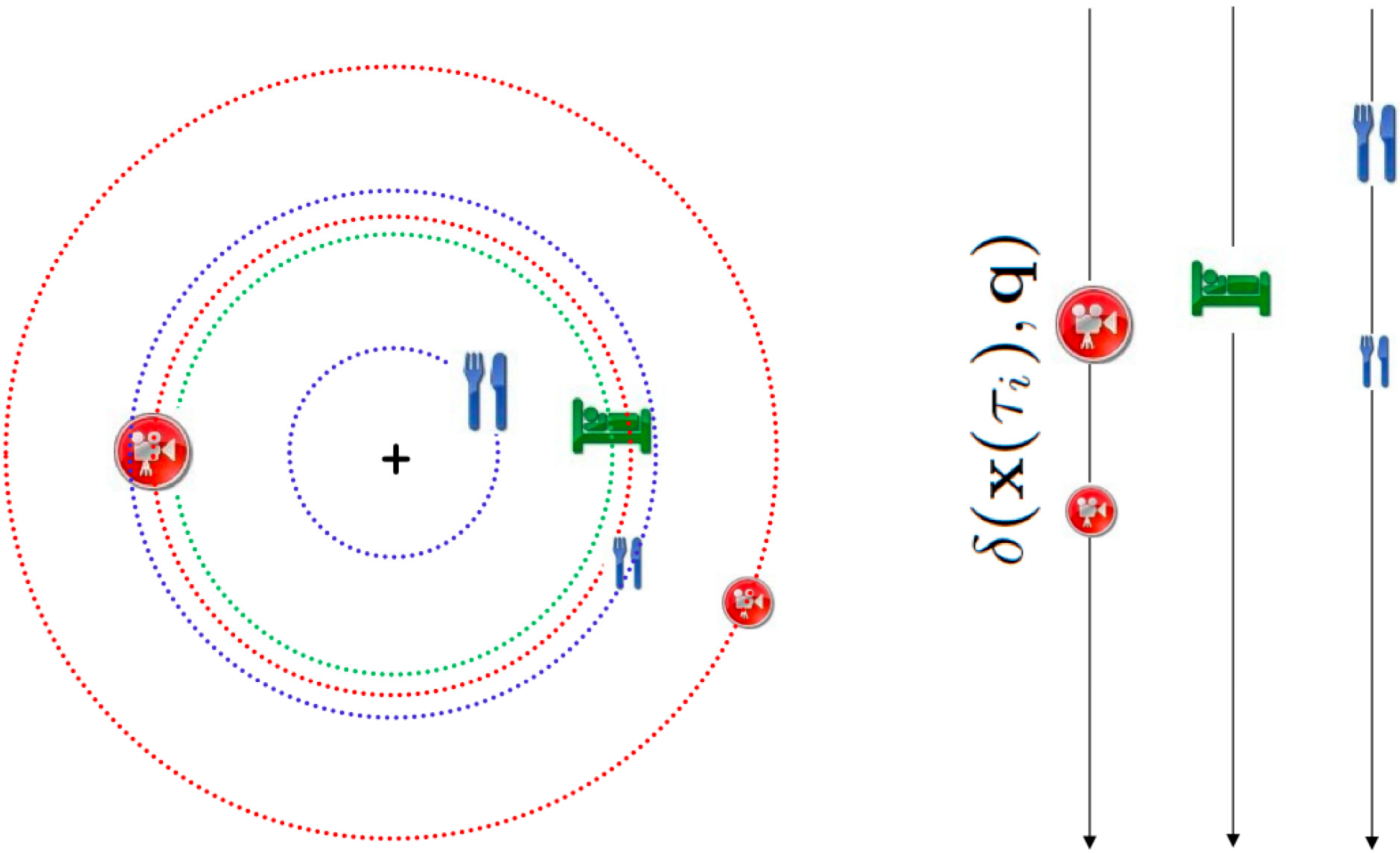




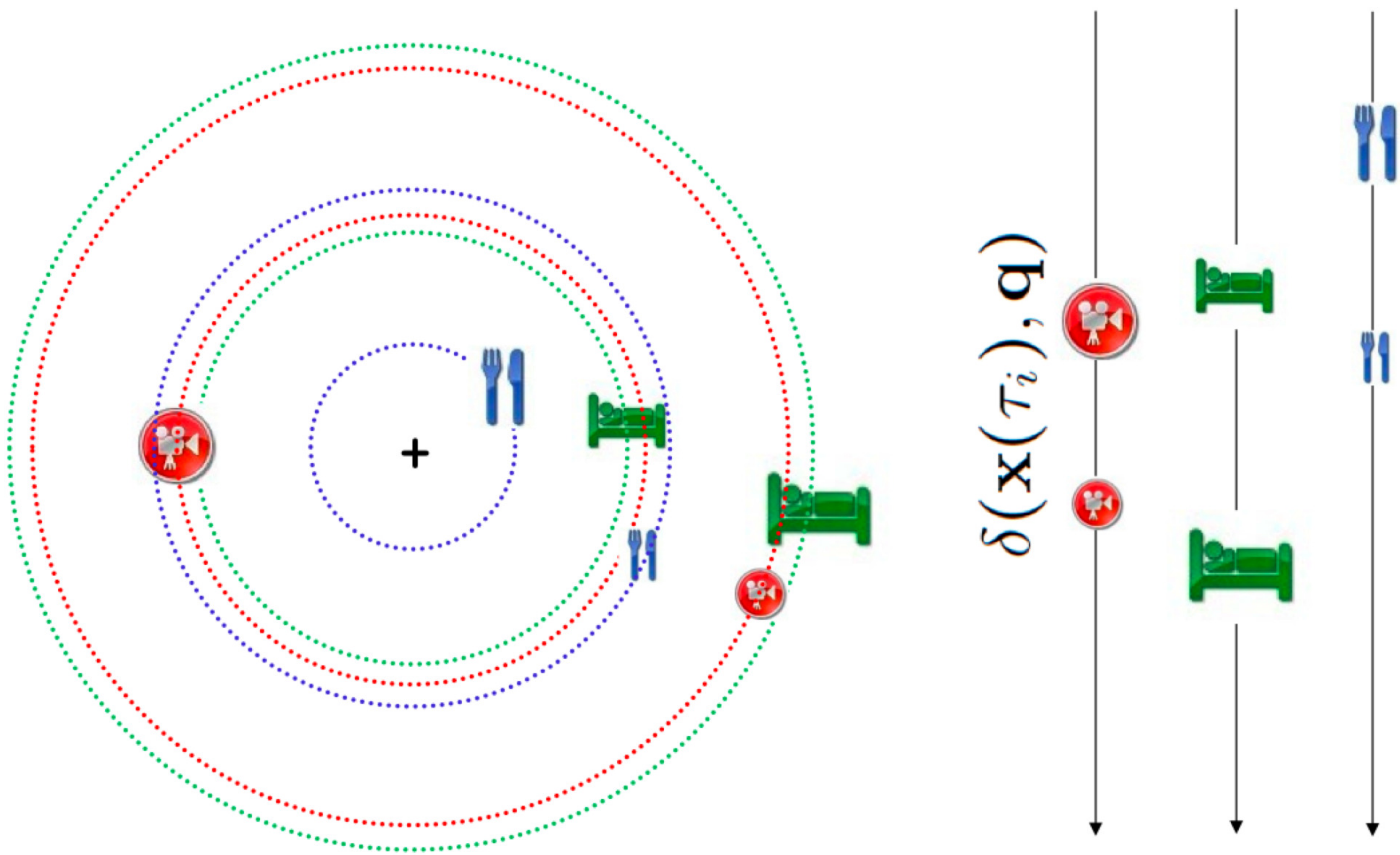
Distance-based access



Distance-based access



Distance-based access



Proximity rank join problems

- Broad applicability
  - Information retrieval
    - E.g. finding similar documents in different collections given a set of keywords
  - Multimedia databases
    - E.g. requesting similar images from different repositories given a sample image
  - Bioinformatics
    - E.g. discovering orthologous genes from different organisms given a target annotation profile

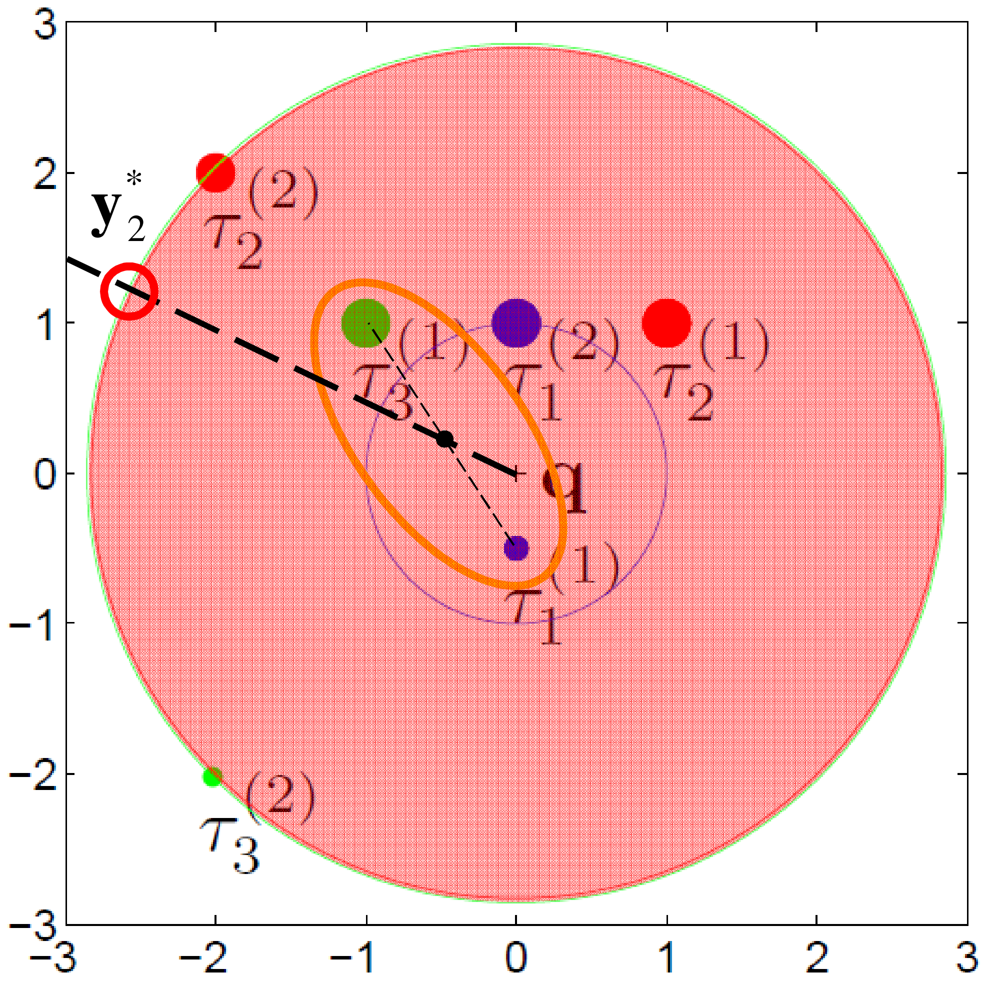
Proximity rank join vs. Rank join

- Proximity rank join closely resembles rank join but…
  - Each tuple is equipped with a feature vector
  - The aggregation function depends on the score and the feature vector
  - The relations are accessed either by distance or by score
- Wanted:
  - Early-out strategy
  - Optimality guarantees (instance optimality)

Bounding scheme

- Stopping criterion based on a bounding scheme:
  - What is the largest aggregate score of a possible combination formed with at least one unseen tuple?
  - We stop when we have k combinations whose score exceeds the bound
- Two options
  - Corner bound (HRJN* – [Ilyas et al., VLDB2004])
    - Fast (but not instance-optimal)
  - Tight bound ([Schnaitter and Polyzotis, PODS2008])
    - Instance-optimal (but slower)
    - Can be computed efficiently when using Euclidean distance

Tight bound for Euclidean distance



M	$\tau \in PC(M)$
$\emptyset$	$\langle\rangle$
$\{1\}$	$\tau_1^{(1)}$ $\tau_1^{(2)}$
$\{2\}$	$\tau_2^{(1)}$ $\tau_2^{(2)}$
$\{3\}$	$\tau_3^{(1)}$ $\tau_3^{(2)}$
$\{1,2\}$	$\tau_1^{(1)} \times \tau_2^{(1)}$ $\tau_1^{(1)} \times \tau_2^{(2)}$ $\tau_1^{(2)} \times \tau_2^{(1)}$ $\tau_1^{(2)} \times \tau_2^{(2)}$
$\{1,3\}$	$\boxed{\tau_1^{(1)} \times \tau_3^{(1)}}$ $\tau_1^{(1)} \times \tau_3^{(2)}$ $\tau_1^{(2)} \times \tau_3^{(1)}$ $\tau_1^{(2)} \times \tau_3^{(2)}$
$\{2,3\}$	$\tau_2^{(1)} \times \tau_3^{(1)}$ $\tau_2^{(1)} \times \tau_3^{(2)}$ $\tau_2^{(2)} \times \tau_3^{(1)}$ $\tau_2^{(2)} \times \tau_3^{(2)}$

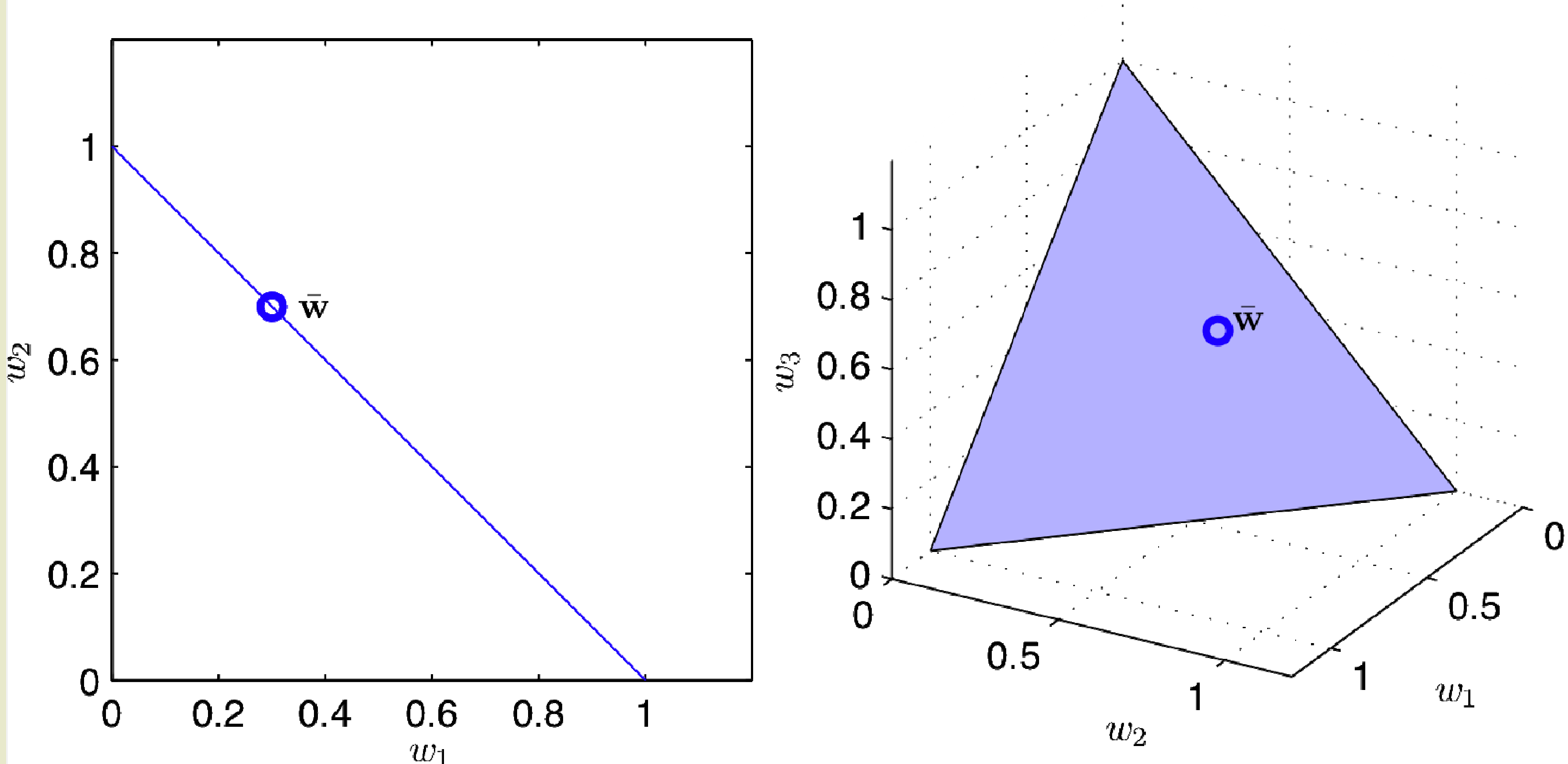
Uncertainty in rank join

- Users are often unable to precisely specify the scoring function
- Using trial-and-error or machine learning may be tedious and time consuming
- Even when the function is known, it is crucial to analyze the sensitivity of the computed ordering wrt. changes in the function

Uncertain scoring

- Assumptions:
  - Linear scoring function
    - $S = w_1 s_1 + w_2 s_2 + \ldots + w_n s_n$
  - User-defined weights $w_1, w_2, \ldots, w_n$ are
    - Uncertain, and, w.l.o.g.,
    - normalized to sum up to 1

Representing scoring functions on the simplex



- Each point on the simplex represents a possible scoring function
- We assume that p(**w**) is uniform over the simplex

Uncertain scoring

- Uncertainty induces a probability distribution on a set of possible orderings
- Each ordering occurs with a probability

$$p(\boldsymbol{\lambda}_N) = \int_{\mathbf{w}\in\Delta^{d-1}, \mathcal{O} \overset{\mathbf{w}}{\leadsto} \boldsymbol{\lambda}_N} p(\mathbf{w})d\mathbf{w}$$

(weights in the simplex inducing that ordering)

- When N is large, we usually focus on a prefix of length K<N of an ordering

Example

- Top-k query:

SELECT R.RestName, R.Street, H.HotelName
FROM RestaurantsInParis R, HotelsInParis H
WHERE distance(R.coordinates, H.coordinates) $\leq 500m$
RANK BY $w_R \cdot$ R.Rating $+ \; w_H \cdot$ H.Stars
LIMIT 5

- Results and possible orderings:

ID	rating	stars
τ_1	2	6
τ_2	7	5
τ_3	4	7
τ_4	5	2

Join Results

Rank By $w_R.rating+w_H.stars$

$w_R+w_H=1$

λ^1	λ^2	λ^3	λ^4	λ^5
τ_3	τ_3	τ_2	τ_2	τ_2
τ_1	τ_2	τ_3	τ_3	τ_4
τ_2	τ_1	τ_1	τ_4	τ_3
τ_4	τ_4	τ_4	τ_1	τ_1

w_R: 0 — 0.167 — 0.4 — 0.571 — 0.833 — 1.0

Representative orderings

- Finding a representative ordering:
  - Most Probable Ordering:

$$\boldsymbol{\lambda}^*_{MPO} = arg. \max_{\boldsymbol{\lambda} \in \Lambda_K} p(\boldsymbol{\lambda})$$

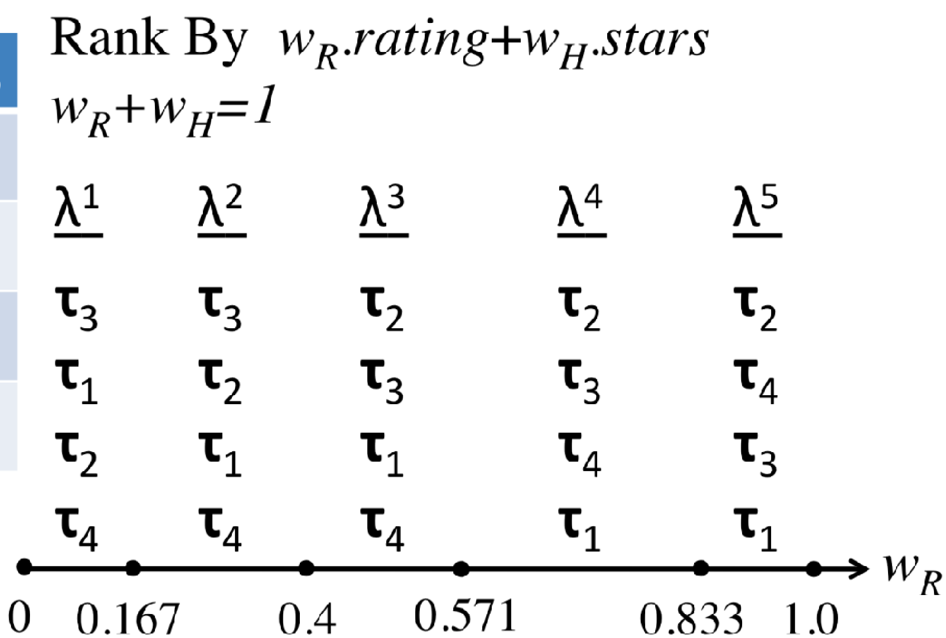
  - Optimal Rank Aggregation:
    - Ordering with the minimum average distance to all other orderings
  - Common distances between orderings:
    - Kendall tau: number of pairwise disagreements in the relative order of items
    - Spearman's footrule: sum of distances between the ranks of the same item in the two orderings

Example of MPO and ORA

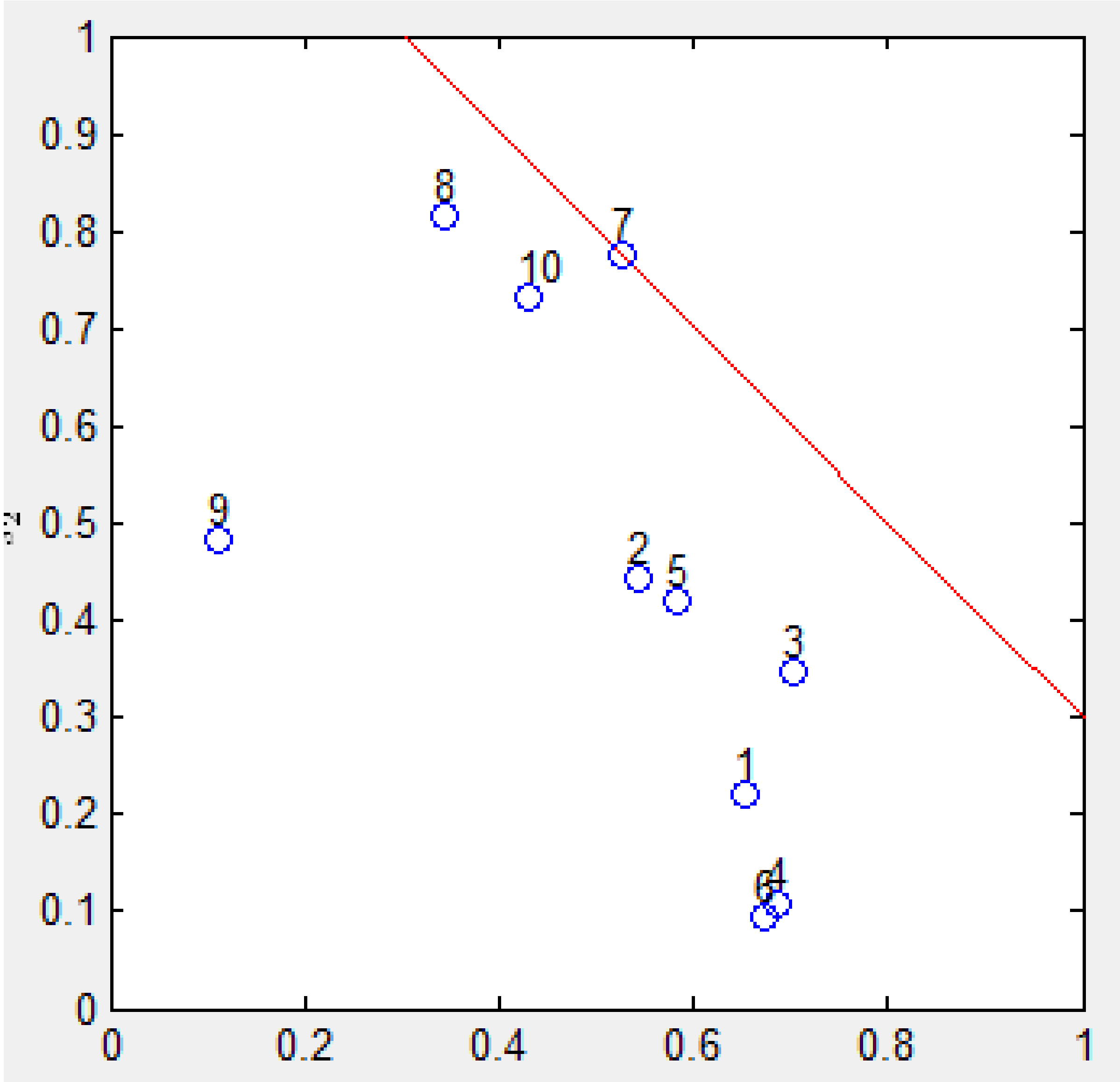
- For K=2, the MPO is $<\tau_2, \tau_3>$
- ORA is λ^3 both for Kendall tau and footrule

ID	rating	stars
τ_1	2	6
τ_2	7	5
τ_3	4	7
τ_4	5	2

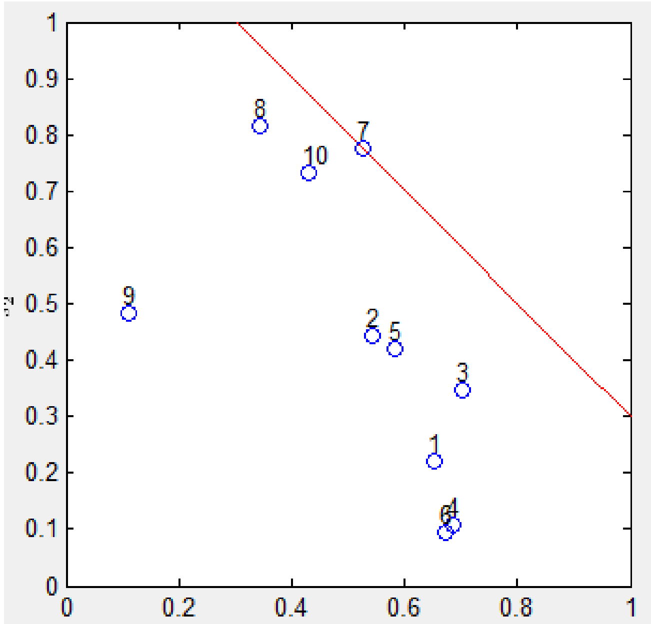
Join Results

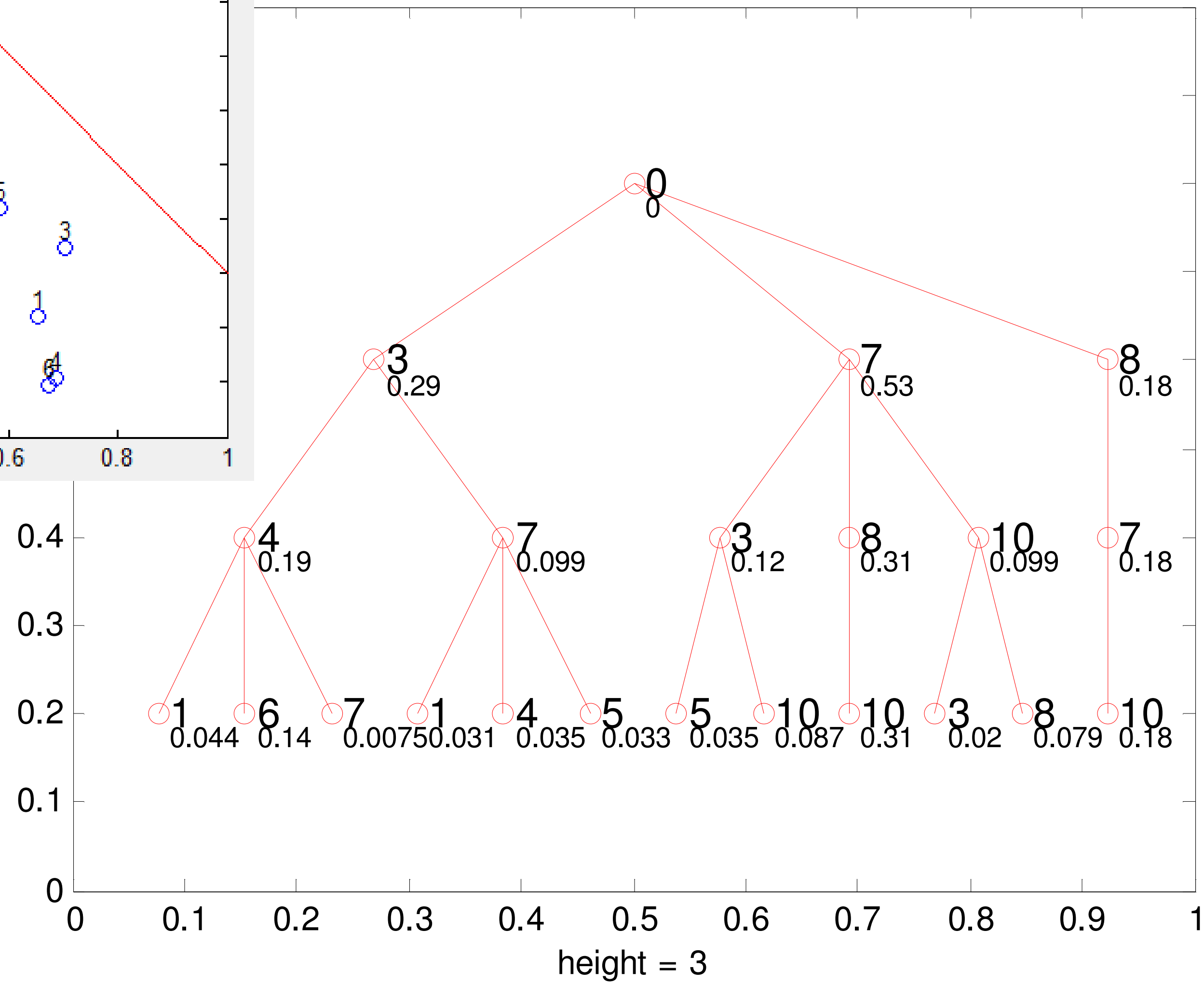


Join results and uncertain scoring function (d=2)



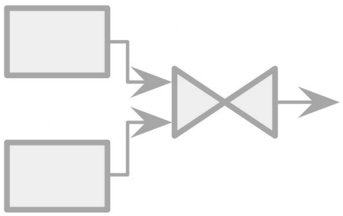
Constructing the possible orderings



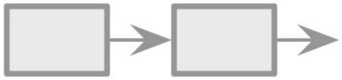


Conclusions

- Rank aggregation
  - Merging ranked lists of objects into a consensus list
- Rank join
  - Extension to heterogeneous (joinable) relations
  - Requires sorted access to data (or even random access)
  - Efficiency measured as total depth (aiming at instance optimality)
- Extensions
  - In proximity r. j. objects are in a vector space affecting the score
  - With uncertain scoring, we look for representative orderings
- More
  - Taking access costs (such as response time) into account
  - Topology of the join



parallel join



pipe join