Efficient Aggregation of Ranked Inputs

Nikos Mamoulis
Department of Computer Science
University of Hong Kong

joint work with
KitHung Cheng, ManLung Yiu, David Cheung

Top-k query

- Combines a number of different rankings of the same set of objects
- Returns the k objects with the highest combined score, based on an aggregate function γ.
- Example:
  - Different web servers provide rankings for the same set of restaurants based on
    - Price
    - Quality
    - Closeness to a particular hotel
  - We want to retrieve the k=10 restaurants with the highest average score

Top-k query example

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

- D = {a,b,c,d,e}
- γ = sum
- top-1 result is object b with score 2.2

Top-k query variants

- The ranked inputs of the query could reside in the same or different servers (centralized or distributed data)
- Standalone query or operator in a more complex query plan
- Only top-k objects are required or both top-k objects and their scores
- On-line (non-blocking) top-k retrieval
- Incremental retrieval of objects with highest scores (k is not predefined)
- Top-k joins (join key is not object-id)
- Probabilistic/approximate top-k retrieval
- Random/sorted accesses at ranked inputs

Our focus

- We focus on top-k aggregation of ranked inputs, for the case where the sorted inputs are accessed only sequentially
  - Random accesses are usually very expensive compared to sorted ones, or restricted/impossible (e.g., web services)
  - May want to merge (possibly unbounded) streams of ranked inputs, produced incrementally and/or on-demand, where individual scores of random objects are not available at anytime.

Our contributions

- We bring to light some key observations, which impose two phases that any top-k algorithm, based on sorted accesses, should go through.
- Based on them, we propose a new algorithm, which is designed to minimize the number of object accesses, the computational cost, and the memory requirements of top-k search.
- We propose adaptations of our algorithm for search variants (exact scores, on-line and incremental search, top-k joins, various aggregate functions, etc.)
- We demonstrate by experimentation that, compared to previous techniques, our method accesses fewer objects, while being orders of magnitude faster.

Previous work: top-k retrieval based on no random accesses (NRA)

- Iteratively retrieves objects and their atomic scores from the ranked inputs in a round-robin fashion.
- For each object x seen so far at any input maintain:
  - γ_x^{ub} : upper bound for x's aggregate score (γ_x)
    - assume that for every S_i, where x has not been seen yet, x's score in S_i is the highest possible (i.e., the score l_i of the last object seen in S_i)
  - γ_x^{lb} : lower bound for x's aggregate score (γ_x)
    - assume that for every S_i, where x has not been seen yet, x's score in S_i is the lowest possible (i.e., 0 if scores range from 0 to 1)
- Let W_k be the set of the k objects with the largest γ^{lb}. If the smallest lower bound in W_k is at least the largest γ_x^{ub} of any object x not in W_k, then W_k is reported as the top-k result and the algorithm terminates.

Example of NRA (k=1, γ=sum)

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

accessed data

- $\gamma_c^{ub} = 2.7$, $\gamma_c^{lb} = 1.8$
- $\gamma_a^{ub} = 2.7$, $\gamma_a^{lb} = 0.9$
- $W_k = \{c\}$
- Since $\gamma_c^{lb} < \gamma_a^{ub}$, we proceed to another round of accesses

Example of NRA (k=1, γ=sum)

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

accessed data

- $\gamma_c^{ub} = 2.6$, $\gamma_c^{lb} = 1.8$
- $\gamma_a^{ub} = 2.6$, $\gamma_a^{lb} = 1.8$
- $\gamma_d^{ub} = 2.5$, $\gamma_d^{lb} = 0.8$
- $\gamma_b^{ub} = 2.5$, $\gamma_b^{lb} = 0.8$
- $W_k = \{c\}$
- Since $\gamma_c^{lb} < \gamma_a^{ub}$, we proceed to another round of accesses

Example of NRA (k=1, γ=sum)

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

accessed data

- $\gamma_c^{ub} = 2.4$, $\gamma_c^{lb} = 1.8$
- $\gamma_a^{ub} = 2.4$, $\gamma_a^{lb} = 1.8$
- $\gamma_d^{ub} = 2.2$, $\gamma_d^{lb} = 0.8$
- $\gamma_b^{ub} = \gamma_b^{lb} = \gamma_b = 2.2$
- $\gamma_e^{ub} = 2.0$, $\gamma_e^{lb} = 0.6$
- $W_k = \{b\}$
- Since $\gamma_b^{lb} < \gamma_a^{ub}$, we proceed to another round of accesses

Example of NRA (k=1, γ=sum)

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

accessed data

- $\gamma_c^{ub} = 2.2$, $\gamma_c^{lb} = 1.8$
- $\gamma_a^{ub} = 2.1$, $\gamma_a^{lb} = 1.8$
- $\gamma_d^{ub} = \gamma_d^{lb} = \gamma_d = 1.8$
- $\gamma_b^{ub} = \gamma_b^{lb} = \gamma_b = 2.2$
- $\gamma_e^{ub} = 1.5$, $\gamma_e^{lb} = 0.9$
- $W_k = \{b\}$
- $\gamma_b^{lb} \geq \gamma_c^{ub}$, thus NRA terminates and reports W_k

Motivation

- NRA accesses objects sequentially from all inputs and updates the upper bounds for all objects seen so far unconditionally.
  - Cost: O(n) per access (the expected distinct number of objects accessed so far is O(n))
  - Space: O(n)
  - No input stream is pruned until the algorithm terminates
- Can we minimize (i) the operations performed at every access, (ii) the memory requirements, and (iii) the number of accesses?

Observation 1

- Let t be the k-th highest score in W_k and $T = \gamma(l_1, \ldots, l_m)$ be the score derived when applying the aggregation function to the last atomic scores seen at each input.
- If t < T, objects which have not been seen so far at any input can end up in the top-k result

Example of Observation 1 (k=1)

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
.	.	.
.	.	.
.	.	.

accessed data

- $T = 2.5$, $W_k = \{c\}$, $t = \gamma_c^{lb} = 1.8$
- There could be an object x not seen yet at any input with scores (0.8, 0.8, 0.9)

Observation 2

- If t < T, any of the objects seen so far can end up in the top-k result.
- Example:
  - T = 2.5, W_k = {c}, t = γ_c^{lb} = 1.8
  - any object {a,b,c,d} could end up in the result

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
.	.	.
.	.	.
.	.	.

accessed data

Implication of observations 1,2

- While t < T the set of candidate top-k objects can only grow and there is nothing we can do about it. Objects which have not been seen so far at any input can end up in the top-k result
- Thus, while t < T, *we should only update* W_k *and* T *while accessing objects from the sources and need not apply expensive updates an comparisons using upper bounds*.

Observation 3

- If t ≥ T, no object which has not been seen at any input can end up in the top-k result.
- Example:
  - T = 2.0, W_k = {b}, t = γ_b^{lb} = 2.2
  - no object x never seen so far can end up in the result

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
.	.	.
.	.	.

accessed data

Implication of observation 3

- As soon as $t \geq T$ the set of candidate top-k objects can only shrink and there is no need to process any newly seen objects.
- Summarizing, observations 1 through 3 imply two phases that all NRA algorithms go through;
  - a *growing* phase during which $T < t$ and the set of top-k candidates can only grow and
  - a *shrinking* phase during which $t \geq T$ and the set of candidate objects can only shrink, until the top-k result is finalized.

Corollary for pruning inputs

- If $t \geq T$ and all current candidate objects have already been seen accessed from input S_i, no further accesses at S_i are required in order to compute the top-k result.

LARA: An efficient top-k algorithm

- LARA: LAttice-based Rank Aggregation
- Based on discussed observations
- Operates differently in the two (growing, shrinking) phases
- Takes its name from the lattice used in the shrinking phase
- Extendable to various top-k query variants

LARA: Growing phase

- While (t<T) // growing phase
  - Iteratively retrieve objects and their atomic scores from the ranked inputs in a round-robin fashion.
  - Update lower bound (γ_x^{lb}) for each newly accessed object x.
  - Update W_k (the set of the k objects with the largest γ^{lb}) and compute t from it.
  - Update T from the last atomic scores seen at each input.
- W_k is updated at O(logk) cost after each access (heap implementation). T's update cost is O(1).

LARA: Shrinking phase

- As soon as $t \geq T$, LARA enters the shrinking phase.
- Objects that have not been seen during the growing phase are immediately pruned (observation 3)
- Instead of explicitly maintaining γ_x^{ub} for each candidate x, LARA reduces the computations based on the following idea.
  - For every combination v in the powerset of m inputs {S1, . . . , Sm}, we keep track of the object x^v in C such that
    - (i) x^v has been seen exactly in the v inputs,
    - (ii) $x^v \notin$ Wk, and
    - (iii) x^v has the highest partial aggregate score among all objects that satisfy (i) and (ii).
  - If $\gamma^{ub}(x^v) \leq t$ we can immediately conclude that no candidate seen exactly in the v inputs may end up in the result.
  - Thus, by maintaining the set of x^v objects, one for each combination v, we can check the termination condition by performing only a small number of $O(2^m)$ comparisons in the virtual lattice.

Example of LARA (k=1)

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

accessed data

- $\gamma_c^{lb} = 1.8$, $\gamma_a^{lb} = 0.9$
- $W_k = \{c\}$, $t = 1.8$, $T = 2.7$, $t < T$
- Still in growing phase

Example of LARA (k=1)

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

accessed data

- $\gamma_c^{lb} = 1.8$, $\gamma_a^{lb} = 1.8$, $\gamma_d^{lb} = 0.8$, $\gamma_b^{lb} = 0.8$
- $W_k = \{c\}$, $t = 1.8$, $T = 2.5$, $t < T$
- Still in growing phase

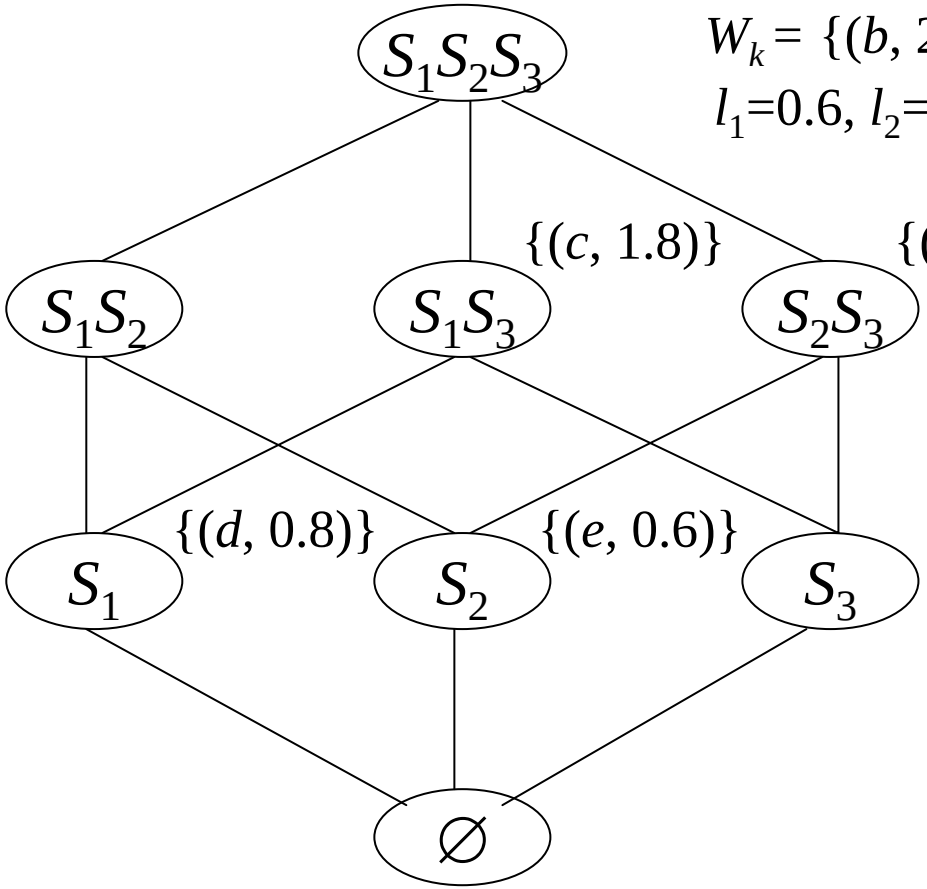
Example of LARA (k=1)

S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

accessed data

- $\gamma_c^{lb} = 1.8$, $\gamma_a^{lb} = 1.8$, $\gamma_d^{lb} = 0.8$, $\gamma_b^{lb} = 2.2$, $\gamma_e^{lb} = 0.6$
- $W_k = \{b\}$, $t = 2.2$, $T = 2.0$, $t > T$
- Entering shrinking phase

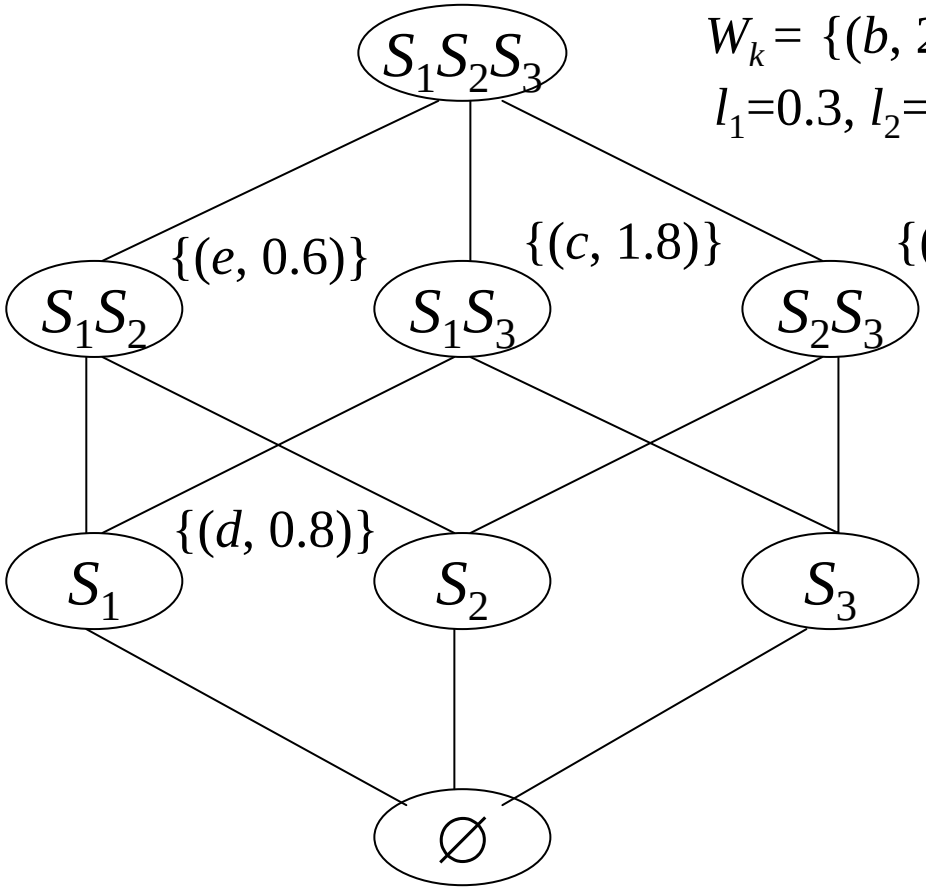
Example of LARA (k=1)



S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

γ_c^{ub} = 2.4 is the greatest upper bound

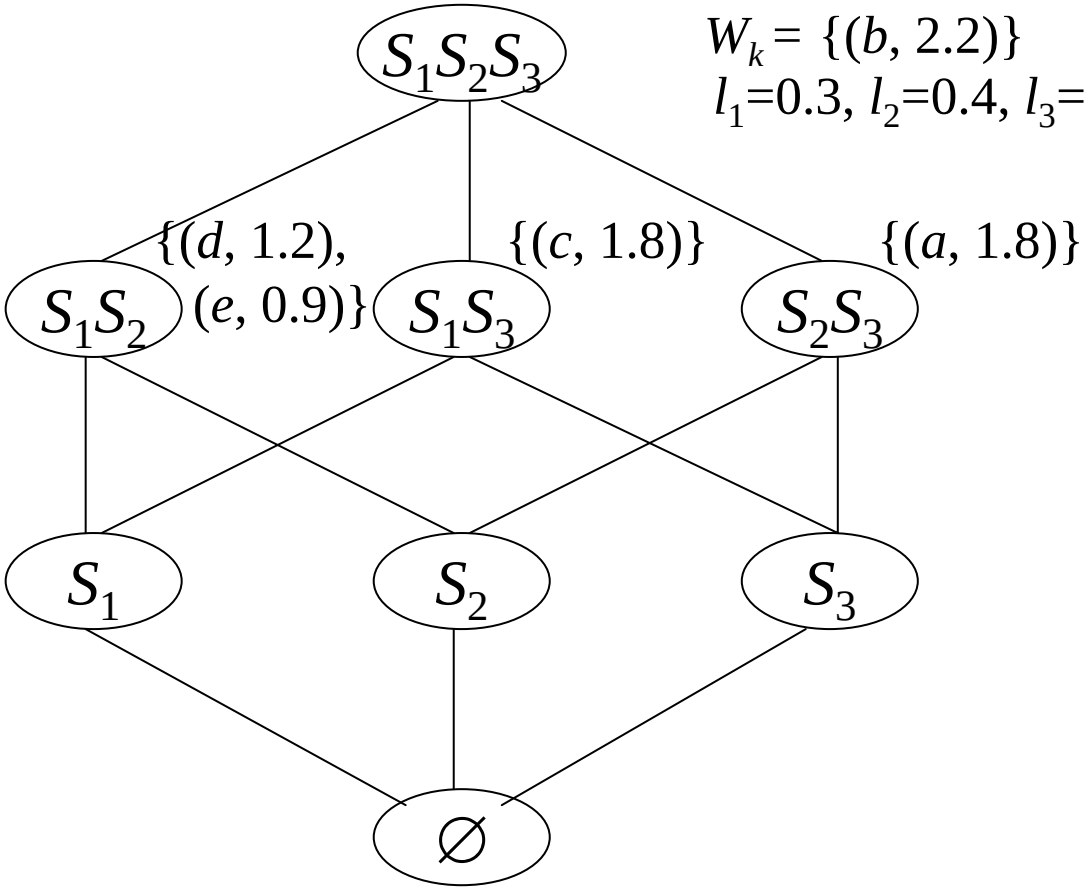
Example of LARA (k=1)



S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

γ_c^{ub} = 2.4 is the greatest upper bound

Example of LARA (k=1)



S1	S2	S3
c 0.9	a 0.9	c 0.9
d 0.8	b 0.8	a 0.9
b 0.6	e 0.6	b 0.8
e 0.3	d 0.4	d 0.6
a 0.1	c 0.2	e 0.5

γ_c^{ub} = 2.2 is the greatest upper bound → LARA stops!

LARA: Computational cost analysis

- Growing phase:
  - O(logk) per access
- Shrinking phase:
  - $O(2^m + \log k)$ per access
- Computational cost is much lower than the O(n) cost of NRA

LARA: Optimizations

- Pruning of candidates
  - If an accessed object may not end up in the top-k result it is immediately pruned
  - Pruning of lattice nodes for which $\gamma^{ub}(x^v) \leq t$
- Reducing the number of comparisons
  - Delay checking for termination if t is much smaller than the greatest $\gamma^{ub}(x^v)$
- Reducing the number of accesses
  - Check for inputs that can be pruned

LARA: Application to top-k variants

- Retrieving exact scores of top-k results
  - LARA-EX: Once the top-k set has been finalized, resume accesses until the complete scores of the results are retrieved
- On-line and incremental search
  - LARA-OL: Return the object with the top score as soon as it is higher than the greatest $\gamma^{ub}(x^v)$
  - LARA-IN: W_k is replaced by a single top object, which is returned next as soon as it is better than the greatest $\gamma^{ub}(x^v)$. Then the algorithm returns to the growing phase (if applicable) and proceeds to finding the next object.
- Various aggregate functions
  - Additional optimizations for the min aggregate function
  - Direct application for weighted and mixed functions

LARA: Application to top-k variants

- Top-k join queries
  - Example:

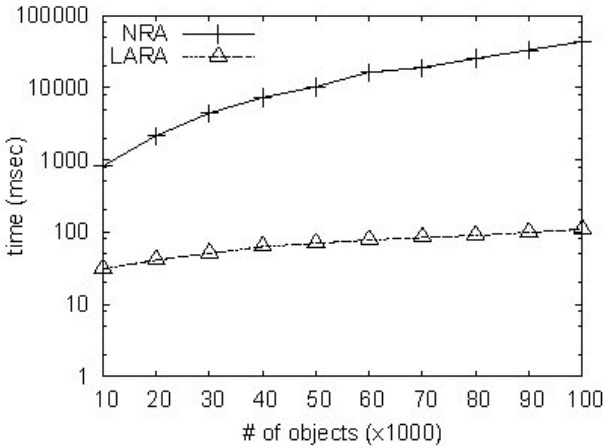
```
SELECT R.id, S.id, T.id
FROM R, S, T
WHERE R.a = S.a
AND S.b = T.b
ORDER BY R.score + S.score + T.score
STOP AFTER k;
```

  - LARA-J: Materializes the lattice and maintains partial join results. When a new tuple is read, it is immediately joined with combinations in the lattice nodes that do not include its source. Partial results are indexed by hash tables to facilitate efficient probing
  - e.g., tuples from R that match with tuples from S are stored in node R ◦ S of the lattice
  - Combinations in the top lattice node (e.g., R◦S◦T) are organized in a priority queue based on their aggregate score and output incrementally, as soon as they are known to have greater score than the greatest $\gamma^{ub}(x^v)$.

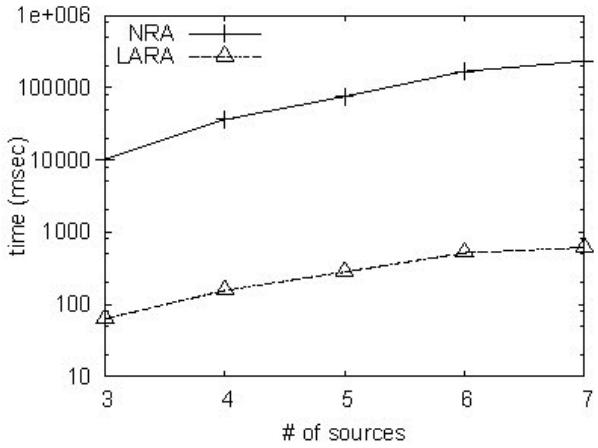
Experimental Settings

- Synthetic data (default 100,000 objects, m=3 inputs):
  - UI: Random and independent scores at individual inputs
  - CO: Correlated scores between inputs
  - AC: Anti-correlated scores between inputs
- Real data from the from the UCI KDD Archive:
  - FC: 581,012 forest coverage cells described by different variables (horizontal and vertical distance to hydrology, distance to roadways, and distance to fire points)
  - CE: 84,443 unweighted Public Use Microdata Series (PUMS) census tuples. Each object in CE is a person (or household) characterized by attributes such as age, rent, wages, number of working hours last week, and number of working weeks last year.

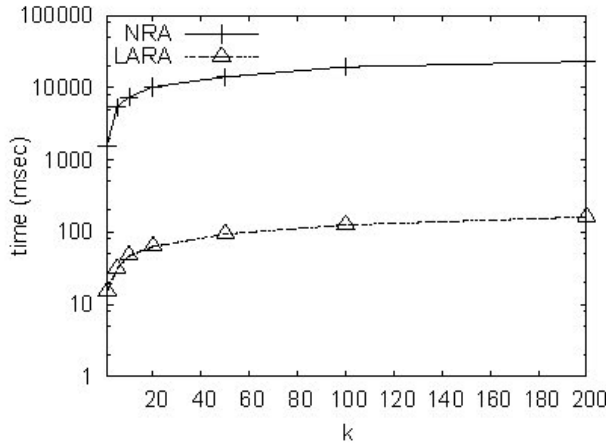
Experimental Comparison (synth. data)



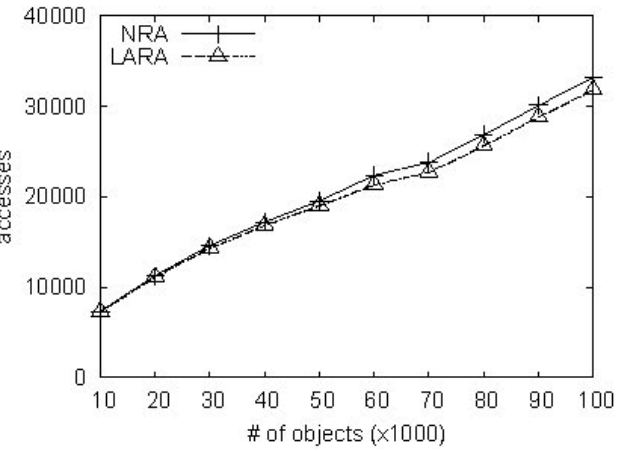
(a) UI, $m = 3, k = 20$



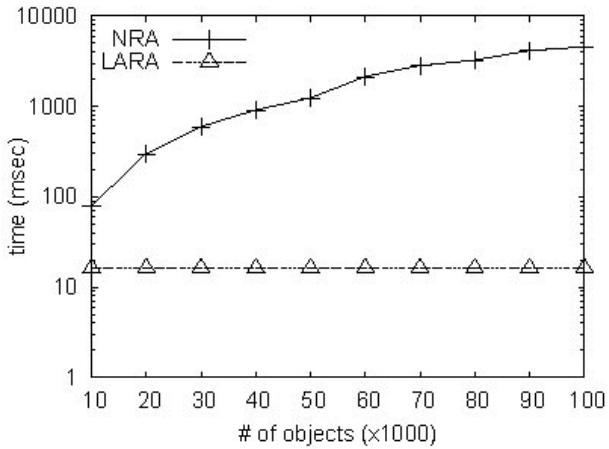
(b) UI, $n = 50\text{K}, k = 20$



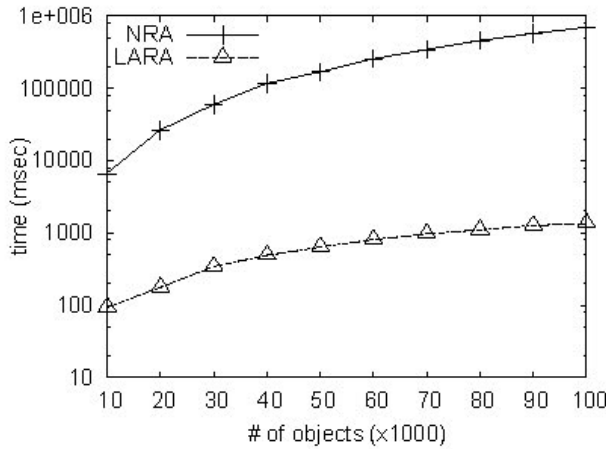
(c) UI, $m = 3, n = 50\text{K}$



(d) UI, $m = 3, k = 20$



(e) CO, $m = 3, k = 20$



(f) AC, $m = 3, k = 20$

Experimental Comparison (real data)

attributes of FC	NRA	LARA
{*hh, hr, hf*}	110529	102168
{*hh, vh, hr*}	358611	217054
{*hh, vh, hf*}	536097	393915
{*hh, hr, hf, vh*}	501231	315042

(a) number of accesses

attributes of FC	NRA	LARA
{*hh, hr, hf*}	517 (377)	0.25 (0.15)
{*hh, vh, hr*}	1047 (471)	0.5 (0.18)
{*hh, vh, hf*}	1614 (703)	0.8 (0.3)
{*hh, hr, hf, vh*}	3415 (1412)	1.06 (0.48)

(b) time in seconds

Figure 6. Forest coverage ($\gamma = \mathtt{sum}$, $k = 20$)

attributes of CE	NRA	LARA
{*r, wh, wy, w*}	171187	161880
{*a, wh, wy, w*}	186694	185962
{*w, wh, wy*}	128390	107258
{*a, r, wh, wy, w*}	238182	237176

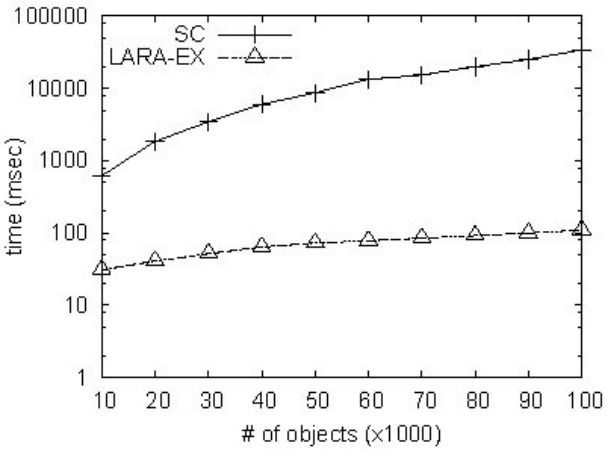
(a) number of accesses

attributes of CE	NRA	LARA
{*r, wh, wy, w*}	300 (227)	0.4 (0.25)
{*a, wh, wy, w*}	299 (228)	0.4 (0.25)
{*w, wh, wy*}	224(122)	0.25 (0.18)
{*a, r, wh, wy, w*}	432 (276)	0.56 (0.32)

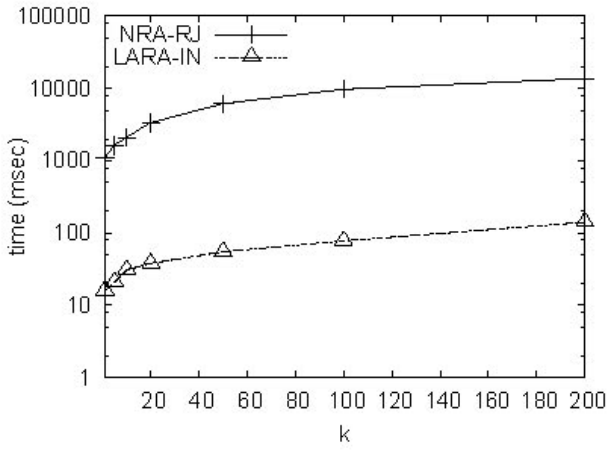
(b) time in seconds

Figure 7. Census data ($\gamma = \mathtt{sum}$, $k = 20$)

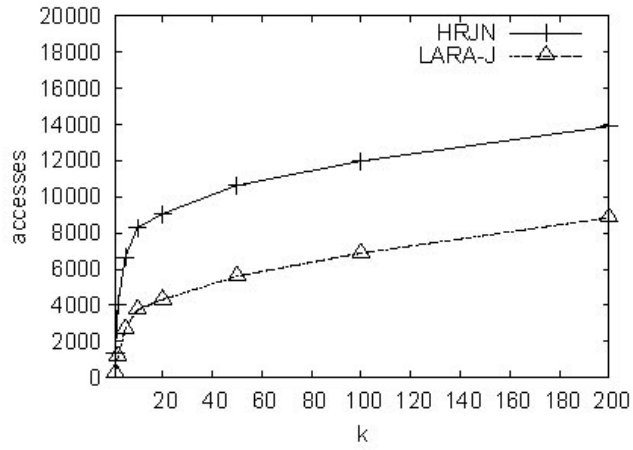
Experimental Comparison (topk variants)



(a) exact scores (UI, $m = 3, k = 20$)



(b) incremental top-k (UI, $n = 50\text{K}, m = 3$)



(c) top-k joins, (UI, $n = 50\text{K}, m = 3$)

Figure 8. Variants of top-k search ($\gamma = \text{sum}$)

γ = min

Forest coverage data

attributes of FC	NRA	LARA	LARA-OPT
{*hh, hr, hf*}	56582	56582	54356
{*hh, vh, hr*}	100579	100579	100579
{*hh, vh, hf*}	113663	113663	86023
{*hh, hr, hf, vh*}	240196	240196	210315

(a) number of accesses

attributes of FC	NRA	LARA	LARA-OPT
{*hh, hr, hf*}	112 (92)	0.22 (0.16)	0.22 (0.16)
{*hh, vh, hr*}	189 (152)	0.36 (0.29)	0.36 (0.29)
{*hh, vh, hf*}	211 (179)	0.47 (32)	0.39 (0.3)
{*hh, hr, hf, vh*}	654 (412)	1.1 (0.72)	1.1 (0.72)

(b) time in seconds

Conclusions

- We developed an efficient ‘no random accesses’ top-k operator, based on some core observations that allow the application of certain optimizations
- The main feature of our method is its low computational cost (orders of magnitude faster than previous methods)
- LARA is also no worse and sometimes significantly better than NRA in terms of object accesses and space requirements
- We showed how it can be adapted for several important variants of top-k search