Optimal Top-k Generation of Attribute Combinations based on Ranked Lists

Jiaheng Lu*, Pierre Senellart#, Chunbin Lin*, Xiaoyong Du*, Shan Wang* and Xinxing Chen*

* School of Information and DEKE, MOE, Renmin University of China

Institute Telecom; Telecom ParisTech, France

{jiahenglu, chunbinlin, duyong, swang, xxchen}@ruc.edu.cn, pierre@senellart.com

Motivation and Problem Statement

Forward		Center		Guard	
F_1	F_2	C_1	C_2	G_1	G_2
Juwan Howard	LeBron James	Chris Bosh	Eddy Curry	Dwyane Wade	Terrel Harris
(G01, 9.31)	**(G02, 8.91)**	**(G05, 7.21)**	(G01, 3.81)	**(G02, 6.59)**	(G09, 7.10)
(G07, 9.02)	(G08, 8.07)	**(G02, 6.01)**	(G06, 3.59)	(G03, 6.19)	(G03, 6.01)
(G03, 8.87)	**(G05, 7.54)**	(G06, 5.58)	(G04, 3.21)	(G04, 5.81)	(G04, 3.79)
(G04, 5.02)	(G10, 7.52)	(G10, 5.51)	(G07, 3.03)	**(G05, 4.01)**	(G08, 3.02)
(G11, 4.81)	(G03, 6.14)	(G04, 5.00)	(G09, 2.07)	(G01, 3.38)	(G05, 2.89)
(G08, 4.02)	(G01, 5.05)	(G11, 3.09)	(G11, 1.70)	(G09, 2.25)	(G02, 2.52)
(G06, 4.31)	(G04, 5.01)	(G01, 2.06)	(G10, 1.62)	(G06, 1.52)	(G01, 2.00)
(G05, 3.59)	(G09, 3.34)	(G08, 2.03)	(G02, 1.59)	(G08, 1.51)	(G10, 1.59)
……	……	……	……	……	……
(G09, 2.06)	(G06, 3.01)	(G09, 1.98)	(G08, 1.19)	(G07, 1.00)	(G06, 1.52)

(a) Source data of three groups

40.27(=21.51+18.76) is the largest among the aggregate scores of top-2

$F_1C_1G_1$	$F_1C_1G_2$	$F_1C_2G_1$	$F_1C_2G_2$	**$F_2C_1G_1$**	$F_2C_1G_2$	$F_2C_2G_1$	$F_2C_2G_2$
(G04, 15.83)	(G04, 13.81)	(G01, 16.50)	(G01, 15.12)	(G02, 21.51)	(G05, 17.64)	(G02, 17.09)	(G02, 13.02)
(G05, 14.81)	(G05, 13.69)	(G04, 14.04)	(G07, 12.05)	(G05, 18.76)	(G02, 17.44)	(G04, 14.03)	(G09, 12.51)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

(b) Top-2 aggregate scores for each combination

Goal: select the top-1 combination of athletes according to their best top-2 aggregate scores for games where they played together.

$F_2C_1G_1$ is the best combination of athletes, as the top-2 games are G02 and G05, and 40.27 (= 21.51 + 18.76) is the highest overall score.

Problem Statement (Top-k,m)

Given a set of groups where each group contains multiple attributes and each attribute is associated with a ranked list.
Top-k,m returns **top-k combinations** of attributes which have the highest overall scores over their **top-m match instances** by a monotonic aggregate function.

Essential difference between top-k queries and top-k,m problem

- Top-k query returns the top-k tuples (objects), while Top-k,m query returns the top-k combinations of attributes in groups.
- Top-k,m queries cannot be transformed into a SQL (nested) query, since SQL queries return tuples, while top-k,m returns attribute combinations based on ranked inverted lists.

Top-k,m Algorithms

Access model

- **Sorted accesses**: read the tuple of lists sequentially.
- **Random accesses**: quickly locate tuples whose ID has been seen by sorted access.

Baseline algorithm: ETA

Compute top-m results for each combination (By Threshold Algorithm **TA**) → Calculate aggregate score for each combination → pick the top-k combinations

Shortcoming

it needs to compute top-m results for each combination and reads more inputs than needed

ULA (Upper and Lower bounds Algorithm)

Compute the **upper** and **lower** bounds for each combination → Return the top-k combinations (at least k combinations which meet the **hit-condition**)

Upper Bound

$$\epsilon^{\max} = \begin{cases} \mathcal{F}_2(tScore(T_1^{\epsilon}),\ldots,tScore(T_{m'}^{\epsilon}),\underbrace{\mathcal{T}^{\epsilon},\ldots,\mathcal{T}^{\epsilon}}_{m-m''}) & m'' < m \\ \max\{\underbrace{\mathcal{F}_2(tScore(T_i^{\epsilon}),\ldots,tScore(T_j^{\epsilon}))}_{m}\} & m'' \geqslant m \end{cases}$$

Lower Bound

$$\epsilon^{\min} = \begin{cases} \mathcal{F}_2(tScore(T_1^{\epsilon}),\ldots,tScore(T_{m'}^{\epsilon}),\underbrace{0,\ldots,0}_{m-m'}) & m' < m \\ \max\{\underbrace{\mathcal{F}_2(tScore(T_i^{\epsilon}),\ldots,tScore(T_j^{\epsilon}))}_{m}\} & m' \geqslant m \end{cases}$$

Hit-condition

One combination ϵ should be an answer if there are at least $N_{\text{com}} - k$ (N_{com} s the total number of the combinations) distinct combinations $\epsilon_1,\ldots,\epsilon_{N_{\text{com}}-k}$ such that $\epsilon^{\min} \geqslant \max\{\epsilon_i^{\max} \mid 1 \leqslant i \leqslant N_{\text{com}} - k\}$

Although ULA is more efficient than ETA, ULA needs to compute the lower and upper bounds for each combination, which may be an expensive operation when the number of combinations is large.

Optimized top-k,m algorithm: ULA+

Pruning combinations without computing the bounds

Intuitively, the combinations with lists containing small top tuples are guaranteed not to be part of answers, as their scores are too small. Therefore, we **do not need to take time to compute their accurate upper and lower bounds**.

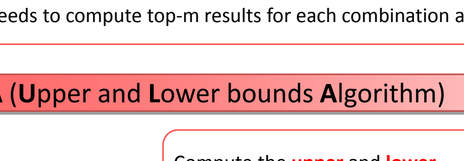
Domination

A combination $\epsilon = \{e_1,\ldots,e_n\}$ is said to dominate another combination $\xi = \{t_1,\ldots,t_n\}$ enoted $\epsilon \gtrsim \xi$ for every 1 > k > n, either holds, $e_i = t_i$ or $L_{e_i} > L_{t_i}$ are two (possibly identical) attributes of the same group G_i.

if ϵ dominates ξ then the upper bound of ϵ s greater than or equal to that of ξ.

Seed combination selection → Dropping useless combinations

Example



1、find the seed combination $\xi = (A_2, B_1)$
2、Drop useless combinations: All combinations (A_i, B_j) $(\forall i \in [3,n], j \in [2,n])$, as well as (A_2, B_j) and (B_1, A_i) are dominated by ξ and can be pruned quickly.

Reducing the number of accesses

Three level optimizations:
- Avoiding both sorted and random accesses for specific lists
- Reducing random accesses across two lists
- Eliminating random accesses for specific tuples

→ Top-k,m Graph (KMG)

Operations in KMG

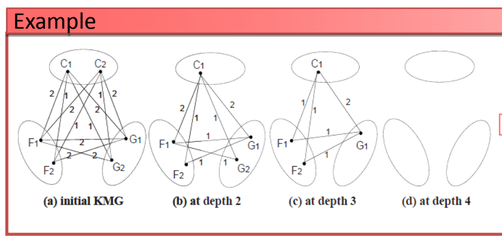
(i) **decreasing the weight** of edges by 1 if one of combinations involving the edge is terminated
(ii) **deleting the edge** if its weight is 0, which means that random accesses between the two lists are useless
(iii) **removing the node** if its degree is 0, which indicates that both sorted and random accesses in this list are useless

Optimized top-k,m algorithm: ULA+

ULA+ Algorithm

Prune dominated combinations → Compute upper and lower bounds for unterminated combinations / Terminate combinations by reducing number of accesses → Return the top-k combinations (at least k combinations which meet the **hit-condition**)

Example



1) Prune the dominated combinations
2) Construct KMG for rest combinations
3) Delete weights, edges and nodes according to operations
4) Compute upper and lower bounds for rest combinations
5) Continue step 3 and 4 until KMG is empty

Optimality properties

Instance Optimality

Let D be the class of all databases. Let A be the class of all algorithms that correctly find top-k,m answers for every database and that do not make wild guesses. If the size of each group is treated as a constant, then ULA and ULA+ are instance-optimal over A and D.

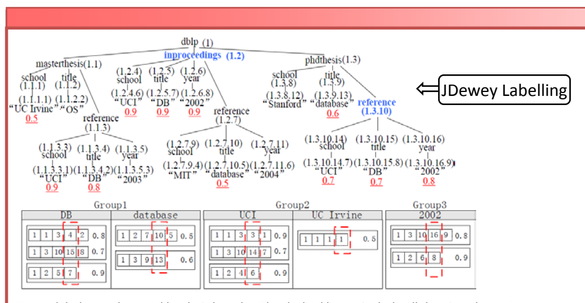
The upper bound of the optimality ratio is tight

There is no deterministic algorithm that is instance-optimal for top-k,m problem, with optimality ratio less than $T + KC_r/C_s$, where $T = \sum_{i=1}^{n} g_i$, $K = \sum_{i \neq j}(g_i g_j)$

No Instance Optimal Algorithms

Let D be the class of all databases. Let A be the class of all algorithms (wild guesses are allowed) that correctly find top-k,m answers for every database. There is no deterministic algorithm that is instance-optimal over A and D.

XML Keyword Refinement



JDewey labels are clustered by their lengths. The dashed boxes include all data in column 4.

Example

- Given a query Q = {DB;UC Irvine; 2002}
- Groups G1 = {"DB"; "database"}, G2={"UCI";"UC Irvine"}, and G3 = {"2002"}.

Consider a top-1,2 query, 17 combinations in total.

Q'={DB, UCI, 2002} with two SLCA nodes (i.e., 1.3.10 and 1.2) The lower bound of Q' is 0.567 + 0.729 = 1.296

Experimental Study

Experimental Setup and Data Sets

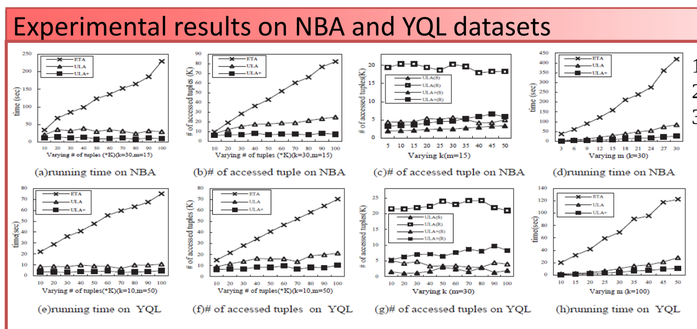
Language: Java; OS: Windows XP; CPU: 2.0GHz; Disk:320GB

Data set	# of objects	# of groups		group size		# of combinaiton	
		max	avg	max	avg	max	avg
YQL	100,100	3	3	150	12	3,375,000	1,728
NBA	31,200	5	5	32	6	33,554,432	7,776
DBLP	3,736,406	7	2.6	12	5	371,292	327

Metrics

(1) running time: the cost of the overall;
(2) access number: the total number of tuples accessed by both sorted access and random access
(3) number of processed combinations: the total number of combinations processed in memory

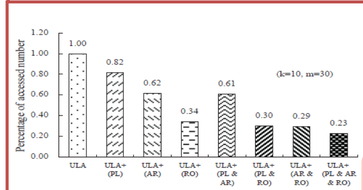
Experimental results on NBA and YQL datasets



(a)running time on NBA (b)# of accessed tuple on NBA (c)# of accessed tuple on NBA (d)running time on NBA
(e)running time on YQL (f)# of accessed tuples on YQL (g)# of accessed tuples on YQL (h)running time on YQL

1、Scalability with database size: Figure (a) (b) (e) (f)
2、Performance vs. range of k: Figure (c) and (g)
3、Performance vs. range of m: Figure (d) and (h)

Both ULA and ULA+ expose an amazingly stable performance while ETA scales linearly. ULA+ outperforms ETA by 1-2 orders of magnitude both in running time and access number.



The performance of different optimizations

ULA+(PL) uses the optimization of "avoiding both sorted and random accesses".
ULA+(AR) applies the optimization of "reducing random accesses across two lists".
ULA+(RO) employs the optimization of "eliminating random accesses for tuples".

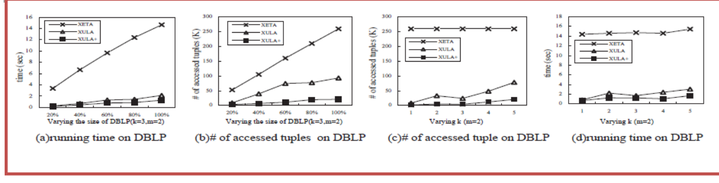
The combination of all optimizations has the most powerful pruning capability.

# of lists in each group	5	10	15	20	25	30
# of total combinations	3125	100000	759375	3200000	9765625	24300000
# of pruned combinations	1875	80000	494325	2332800	7604375	19756800
Pruning Percentage	60.0%	80.0%	65.1%	72.9%	77.9%	81.3%

The performance of optimization to reduce combinations

The number of combinations processed in memory by our optimized algorithm is far less than that of ULA.

Experimental results on DBLP dataset



(a)running time on DBLP (b)# of accessed tuples on DBLP (c)# of accessed tuple on DBLP (d)running time on DBLP

XULA and XULA+ perform better than XETA and scale well in both running time and number of accesses.

Conclusion

- Propose a new problem called top-k,m query evaluation
- Developed a family of efficient algorithms, including ULA and ULA+
- Analyzed different classes of data and access models, such as group size and wild guesses and their implication on the optimality of query evaluation
- Showed how to adapt our algorithms to the context of XML keyword query refinement