Efficient Processing of Exploratory Top-k Joins

Orestis Gkorgkas[1], Akrivi Vlachou[1,2], Christos Doulkeridis[3] and Kjetil Nørvåg[1]
[1]Norwegian University of Science and Technology (NTNU), Trondheim, Norway
[2]Institute for the Management of Information Systems, R.C. "Athena", Athens, Greece
[3]University of Piraeus, Piraeus, Greece
{orestis,vlachou,noervaag}@idi.ntnu.no, cdoulk@unipi.gr

ABSTRACT

In this paper, we address the problem of discovering a ranked set of k distinct main objects combined with additional (accessory) objects that best fit the given preferences. This problem is challenging because it considers object combinations of variable size, where objects are combined only if the combination produces a higher score, and thus becomes more preferable to a user. In this way, users can explore overviews of combinations that are more suited to their preferences than single objects, without the need to explicitly specify which objects should be combined. We model this problem as a rank-join problem where each combination is represented by a set of tuples from different relations and we call the respective query *eXploratory Top-k Join* query. Existing approaches fall short to tackle this problem because they impose a fixed size of combinations, they do not distinguish on combinations based on the main objects or they do not take into account user preferences. We introduce a more efficient bounding scheme that can be used on an adaptation of the rank-join algorithm, which exploits some key properties of our problem and allows earlier termination of query processing. Our experimental evaluation demonstrates the efficiency of the proposed bounding technique.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*

General Terms

Algorithms, Experimentation, Performance

Keywords

Top-k queries, rank-join queries, exploratory queries

1. INTRODUCTION

Top-k queries [4] are often used to help users select the k best objects according to their preferences from a large set of objects. A product is typically represented by a d-dimensional point p where each dimension describes a specific feature. Usually, preferences are expressed through a weighting vector $\mathbf{w}$ of d dimensions, each corresponding to an attribute of the product, while the value of the dimension indicates the importance of the specific attribute to the user. The ranking of the objects is based on a scoring function $f_{\mathbf{w}}(p)$, and one of the most common ones is the weighted sum $f_w(p) = \sum_i^d \mathbf{w}[i]p[i]$.

Mobiles

Model	weight gram.	cam.res storage Mpixel	memory GB	price EUR	ext.mem. type
S5	145	16	16	688	1
I5S	112	8	32	848	0
I5C	132	8	16	614	0
S4	153	8	16	441	1
G2	143	13	32	454	0

Memory

model	Capacity	Price	mem. type
A	64	86	1
B	64	71	1
C	32	37	1

Figure 1: Sample Database

For instance, considering the database of Figure 1, if $\mathbf{w} = \{0.3, 0.1, 0.4, 0.2\}$ is a preference vector of user wishing to buy a mobile phone who is particularly interested in storage space, then the iPhone 5S and the LG G2 are the top-2 results returned to the user. However S4, and S5 can be combined with a memory card and provide the user with a product which is more suited to her needs than a single phone. Taking into account the available combinations the top-2 results are now the combinations between S5 and the memory cards A and B. Although both options are more suited to the user than a single phone, they are quite similar and provide little overview of the available products of her interest (phones). A more convenient approach would be to present to the user only the best combination for each phone and thus providing at the same time combinations that suit the user's needs while maintaining the maximum overview of the products the user is interested in.

To this end, we propose the eXploratory Top-k Join (XTJ_k) query. An XTJ_k takes as input a a set of relations where there is a *main* relation and the rest *additional* relations are joined to the main relation forming a "star"-like structure. Among all possible combinations, only the best for each product are considered and the top-k of them are returned to the user.

Current state-of-the-art techniques [2, 3, 7] for computing combinations based on preference vectors fall short to address this kind of queries as they assume that each result should contain objects from all relations participating in the join. On the contrary, our re-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SSDBM'14, June 30 – July 2, 2014, Aalborg, Denmark.
Copyright 2014 ACM 978-1-4503-2722-0/14/06 ...$15.00.
http://dx.doi.org/10.1145/2618243.2618280 .

quirement is that an object should be added to a combination only if it is beneficial for the combination. Moreover, current techniques do not exploit the form of the result-set and the structure of the join, fact that leads to suboptimal performance.

To summarize, the contributions of this paper are: a) we introduce the eXploratory Top-k Join (*XTJ*$_k$) query, a novel query type which creates combinations of variable size between main and additional objects and returns the top-k combinations with discrete main objects, b) we introduce an efficient bounding scheme for our algorithm, and c) we perform an experimental evaluation that demonstrates the efficiency of our approach.

2. RELATED WORK

Top-k queries have been well-studied in the last years to enable ranked retrieval of objects based on user preferences. For a thorough overview we refer to [4]. Regarding rank-join queries, Ilyas et al. proposed the HRJN algorithm [3] while Schnaitter et al. [7] proposed the a-FRPA algorithm which improves the performance of HRJN* in low dimensionality by using an improved bounding technique. However for more than 4 dimensions the performance advantage is minimal.

Our work is also related to package recommendation [5, 6, 8]. Xie et al. [8] study the problem of creating the best package out of a set of items given a specific budget. However, it is assumed that all objects can be joined with one another and the problem they address is to create the most attracting package of objects for a user.

Our main differences from the aforementioned approaches are that we focus on maximizing the preference score of the main objects rather than creating combinations which satisfy a certain constraint. In addition, we do not assume a static combination size but we consider it to be dynamic. We examine the specific case of joins where all additional objects are combined with a main object. This case of "star"-join has specific characteristics which allow us to improve the performance of calculation of such joins. Finally, we do not assume that all combinations are possible but we evaluate the join conditions at the same time. None of the aforementioned approaches examine all these conditions simultaneously.

3. PROBLEM DEFINITION

In this section we formally define the *XTJ*$_k$ query and all necessary structures used both for the problem definition and the description of the respective algorithms.

3.1 Object Combinations

Let $\mathcal{D}$ be a database of objects and E_M be a relation in $\mathcal{D}$ which is connected to a set of relations $\mathcal{E} = \{E_1, \ldots, E_n\}$ of $\mathcal{D}$. We assume that E_M has a set of d real valued attributes $A_{E_M} = \{a_1, \ldots, a_d\}$ and each relation E_i contains a subset $A_{E_i} \subseteq A_{E_M}$ of the attributes of the main relation. Each object in a relation $E \in \mathcal{E} \bigcup \{E_M\}$ is represented as a d-dimensional point $p \in \mathbb{R}^d$ where $p[i] \in \mathbb{R}$ if $a_i \in A_E$ and $p[i] = 0$ if $a_i \notin A_E$. We refer to E_M as *main relation* and to relations $E_i \in \mathcal{E}$ as *additional relations*. The objects of the relations are referred to as *main* and *additional* objects respectively.

Using the main and the additional objects we can form *combinations* where each combination contains exactly one main object and at most one object from each additional relation. We say that an object of the main entity relation $o \in E_M$ and an object $p \in E_i$ of an additional relation E_i are *combinable* if there is a join of the form $o \bowtie p \in E_M \bowtie E_i$. Given a relation E_M we denote as $C(E_M)$ the set of all possible combinations that can be formed using E_M as main relation.

As an example, using the database of Figure 1, C(Mobiles) consists of all combinations with a mobile a main object and $\{S5\}$, $\{I5S\}$ and $\{S5, A\}$ are valid combinations while $\{I5S, A\}$ and $\{S5, A, B\}$ are not.

3.2 Combination Ranking

We consider a user query to be a d-dimensional preference vector $\mathbf{w}$ targeted to relation E_M and each dimension $\mathbf{w}[i]$ of the query to represent the importance of the respective attribute to the user. Without loss of generality we assume that $\sum_{i=1}^{d} \mathbf{w}[i] = 1$ and if a user is not interested in a specific attribute a_i of the main objects then $\mathbf{w}[i] = 0$.

Given a vector $\mathbf{w}$, a *top-k query* returns the k best objects in E_M according to the scoring function $f_{\mathbf{w}}(o) = \sum_{j=1}^{d} \mathbf{w}[j]o[j]$. We refer to the top-$k$ objects as the $TOP_k(\mathbf{w})$ set of E_M. Similarly, the score of any object p in an additional relation is equal to $f_{\mathbf{w}}(p) = \sum_{j=1}^{d} \mathbf{w}[j]p[j]$. The score of a combination c comes now naturally as $f_{\mathbf{w}}(c) = \sum_{j=1}^{d} \mathbf{w}[j] \sum_{p \in c}(p[j])$.

A combination for which it holds that no other tuple can be added and improve its score is called *total combination*. A total combination does not necessarily contain objects from every additional relation. It is possible that a main product is not combinable with a category of additional products (e.g. I5S is not combinable with memory cards) or the score of an additional object is negative and the addition of such an object to a combination would make the score of the combination worse.

Each object of the main relation can participate in many combinations but for each main object we are interested only in the combination with the best score which we call the *candidate combination*. For each object there is only one candidate combination. We denote the set of all candidate combinations as $B(E_M)$. Obviously $B(E_M) \subseteq C(E_M)$. It is worth noting that a candidate combination is always total. If it were not total then the addition of an additional object would create a better combination which would contradict our claim that the candidate combination is the best possible combination for a specific main object.

We can now extend the top-k query in order to take into account not only single objects but also combinations. We therefore introduce the *eXploratory Top-k Join* (*XTJ*$_k$) query which returns the top-k candidate combinations. In other words, an *XTJ*$_k$ query lists the k main objects with the best combinations and thus each main object can appear maximum once in the query's result-set.

DEFINITION 1. ***XTJ*$_k$ *query.*** *Given a main relation E_M, a set of additional relations $\mathcal{E}$, a preference vector $\mathbf{w}$ and an integer k, the result set XTJ$_k(\mathbf{w})$ of an eXploratory Top-k Join query is a set of candidate combinations such that XTJ$_k(\mathbf{w}) \subseteq B(E_M)$, $|XTJ_k(\mathbf{w})| = k$, $\forall c_1 \in XTJ_k(\mathbf{w}), c_2 \in B(E_M) - XTJ_k(\mathbf{w})$ it holds that $f_{\mathbf{w}}(c_1) \geq f_{\mathbf{w}}(c_2)$.*

Returning to our example, the result-set of the *XTJ*$_2(\mathbf{w})$ query for $\mathbf{w} = (0.3, 0.1, 0.4, 0.2)$ is the set $\{\{S5, B\}, \{S4, B\}\}$. We should note that the combination $\{S5, A\}$ has a better score than $\{S4, B\}$ but the result-set can contain only each main product only once and thus $S5$ can be included only once in the result-set.

4. THE HRJN ALGORITHM

A straightforward approach in order to process *XTJ*$_k$ queries is to adapt an existing rank-join algorithm, namely *HRJN* [3]. We modified HRJN in a way that all possible combinations are computed and not only the ones that include all relations. We employ

the modified HRJN algorithm as a baseline to compare the performance of our algorithms.

Bounding scheme. A rank-join [3] algorithm typically accesses inputs in ranked order based on the query at hand. Such an algorithm estimates the upper bound of the score that any unseen tuple can produce and terminates when the k-th best join result found is better than the upper bound. For the additional relations we include the best tuples only if their score is positive and they can increase the total score of the combination. We must include however the score of the best tuple of the main relation since its presence in the results is necessary. The bounding scheme of HRJN is formally described in Equations 1-3.

$$UB_{E_M} = f_{\mathbf{w}}(HE_M[last]) + \sum_{E \in \mathcal{E}} u(f_{\mathbf{w}}(HE[1])) \quad (1)$$

$$UB_{E_i} = f_{\mathbf{w}}(E_M[1]) + f_{\mathbf{w}}(E_i[last]) + \sum_{\substack{E \in \mathcal{E} \\ E \neq E_i}} u(f_{\mathbf{w}}(E[1])) \quad (2)$$

$$UB_{HRJN} = \max_{E \in \mathcal{E} \bigcup \{E_M\}} (UB_E) \quad (3)$$

The notation HE denotes the set of accessed objects of relation E and by $HE[1], HE[last]$ we denote the first and last accessed tuples. The function $u(x)$ returns x if $x > 0$ and 0 if $x \leq 0$. It can be proved that the adapted version of HRJN provides a correct solution to the Exploratory top-k join problem.

5. EXPLORATORY RANK-JOIN (XRJN)

In this section, we propose the exploratory rank-join algorithm (*XRJN*), which employs an improved bound. Before we continue it is necessary to introduce the notion of the *combination schema*. Intuitively, a combination schema is a set of entity relations that can produce combinations.

DEFINITION 2. ***Combination schema.*** *Given a main relation E_M and a set of additional relations $\mathcal{E}$ a combination schema CS is a set of relations such that $CS \subseteq \mathcal{E} \bigcup \{E_M\}$ and $E_M \in CS$.*

Given a combination we denote the combination schema that the combination belongs to as $CS(c)$.

Bounding scheme. At a random state of the algorithm, let us denote with HE the objects of a relation E that have been accessed so far, and with $\hat{B}(E_M)$ the set of all combinations that have been created so far. Recall that only one combination per main object is created and that an additional object is added to a combination only if this produces a better score for the combination. There are two bounds we should consider, denoted as UB_{E_M} and UB_{comb} respectively, and our bound (UB_{XRJN}) is their maximum:

$$UB_{XRJN} = max(UB_{E_M}, UB_{comb}) \quad (4)$$

The first (UB_{E_M}) is to determine the upper bound of any unseen object of the main relation E_M. In the best case, the next object of the main relation to be accessed will be combined with the best objects of the additional relations except for those that have a negative score. Apparently, the upper bound for any unseen object of the main relation relation is the same with the upper bound calculated by the baseline and its value is calculated by Equation 1.

The second (UB_{comb}) is the upper bound for any unseen object of the additional relations. Let p be the next object of an additional relation E_i to be accessed. We distinguish two cases: The object is possibly combinable either with a main object $o \in E_M$ that has been accessed and therefore $o \in HE_M$ or with an object that has not yet been accessed and therefore $o \in E_M - HE_M$. If $o \in E_M - HE_M$ then the upper bound will include the $HE_M[last]$ and $HE_i[last]$ and therefore it will always be worse than UB_{E_M}. Consequently, we can safely ignore this alternative. The next alternative is that p is combinable with one or more main objects that already participate in a combination. We should remind here that a single object $o \in E_M$ is also a valid combination. The upper bound of this case is determined by the score of the *most promising* combination. The most promising combination is the non-total combination c_{mp} which has the largest score if we add all the last accessed objects for all relations that are not included in the combination schema $CS(c)$. We should note that this bound is the same for all additional relations. Therefore we call it UB_{comb} because it depends on the current combinations and the last objects accessed. The most promising combination and the respective bound UB_{comb} are formally defined in Equations 5 and 6.

$$c_{mp} = \underset{\substack{c \in \hat{B}(E_M) \\ c \text{ not total}}}{argmax} \left(f_{\mathbf{w}}(c) + \sum_{\substack{E \in \mathcal{E} \\ E \notin CS(c)}} u(f_{\mathbf{w}}(HE[last])) \right) \quad (5)$$

$$UB_{comb} = f_{\mathbf{w}}(c_{mp}) + \sum_{\substack{E \in \mathcal{E} \\ E \notin CS(c_{mp})}} u(f_{\mathbf{w}}(HE[last])) \quad (6)$$

6. EXPERIMENTAL EVALUATION

In this section, we present the results of the experimental evaluation. All algorithms were implemented in Java and the experiments run on an AMD Opteron 4130 Processor (2.00GHz), with 32GB of RAM and 2TB of disk. The relations were sorted for each query, stored to the disk and accessed sequentially.

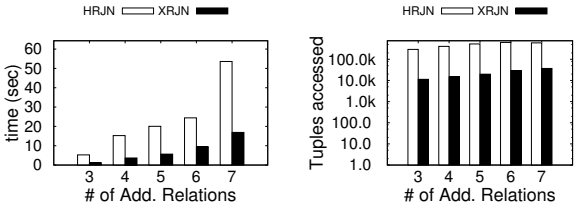
Datasets and metrics. For the data set D, all data values for all relations and dimensions were generated independently using a uniform distribution generator. Each additional relation has a random subset of attributes of the main relation and contains at least two attributes but no more than $d-1$ where d is the number of attributes of the main relation. The combinations of attributes of each additional relation is unique. Each additional relation has also a joining attribute which does not participate in the ranking of each object while the main relation has $|\mathcal{E}|$ joining attributes, one for each additional relation. The values for the joining attributes are decided based on the selectivity value σ. In order to achieve a variation on the joining selectivity, for each additional relation the joining attribute takes values in the space $[0, (\sigma \times \mathrm{r})^{-1})$ where r is a random number in the space $[1, 3]$. Finally the size of each additional relation is equal to the size of the main relation and all attributes were normalized in the interval $[0, 10000]$.

The metrics under which we evaluated the implemented algorithms were: a) execution time required by each algorithm, and b) total tuples accessed (depth).

Experimental procedure. We run a series of experiments varying the parameters of: a) the number of additional relations ($|\mathcal{E}|$) in the interval [3-7], b) dimensionality (d) in the interval [4-8], c) number of returned results (k) in the interval [5-100], and d) selectivity (σ) [0.001-0.05].

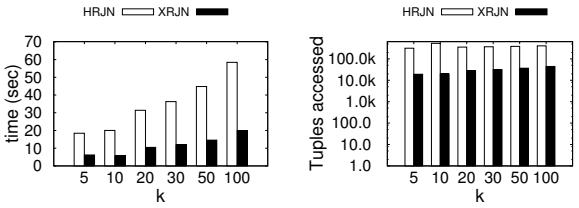
The default setup for the experiments was: $|\mathcal{E}| = 5$, $|E_M| = 100K$, $k = 10$, $\sigma = 0.001$, while the number of preference queries for each setting was equal to $|W| = 100$. Both the dataset and the preferences set followed the uniform distribution.

Varying $|\mathcal{E}|$. As shown in Figures 2-5 the improved XRJN bound is from 2 to 3 times more efficient than HRJN bound. In more detail, Figure 2(b) suggests that XRJN accesses nearly an order



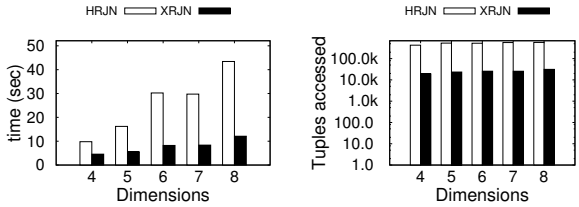
(a) time (b) depth

Figure 2: Varying # of additional relations



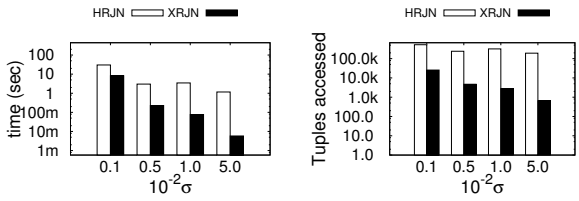
(a) time (b) depth

Figure 3: Varying number of results



(a) time (b) depth

Figure 4: Varying dimensionality



(a) time (b) depth

Figure 5: Varying selectivity

of magnitude less tuples than HRJN while Figure 2(a) indicates that XRJN needs 2-3 times less processing time. The difference in processing time is not as large as the difference in accessed tuples because the XRJN bound requires considering all created combinations while HRJN considers only $2(|\mathcal{E} \bigcup \{E_M\}|)$ tuples. It is worth mentioning that as the number of additional relations increases, the number of accessed tuples raises not only because more relations have to be accessed but also because the upper bound drops slower. As more additional relations are processed, the access cost of the increased number of tuples for HRJN dominates the processing cost of the XRJN bound.

Varying k. Figure 3 illustrates the performance of HRJN and XRJN when varying k. As k raises the upper bound needs to drop more and the lower the bound is, the harder it becomes for the HRJN bound to drop. On the other hand, XRJN, by taking into account the formed combinations and the accessed tuples, manages to reduce the upper bound faster.

Varying dimensionality. Similar conclusions can be drawn from Figure 4. As the preference vectors include more dimensions, the significance of each dimension drops, causing the HRJN bound to drop slowly as simultaneous decrease of all dimensions is needed. XRJN diminishes the problem of high-scored tuples by calculating the upper bound based on the already formed combinations and therefore terminates faster.

Varying join selectivity. The join selectivity is of particular interest as it affects the performance of XRJN significantly. As shown in Figure 5, when join selectivity rises, total combinations are formed much faster and therefore they lead the XRJN bound to decrease. HRJN is less affected by the join selectivity as it does not consider the formed combinations during the bound calculations. The critical role of the formed combinations becomes clear as for selectivity values close to $\sigma = 0.01$, the difference in performance increases significantly and reaches nearly two orders of magnitude with respect to both time and tuples accessed.

7. CONCLUSION

In this paper, we addressed the problem of discovering a set of k combinations between main and additional objects of k with distinct main objects. Our approach tries to balance between finding the most suitable combination of objects for the user while providing to her a wide overview of the available objects. We define the eXploratory Top-k Join query and we present a new bounding technique for efficiently calculating the result. Our experimental evaluation demonstrates that our bounding technique provides a better estimation of the upper bound in comparison to the rank-join bounding technique and therefore it allows the joining algorithm to terminate faster. In our future work, we intend to design scalable algorithms for processing exploratory top-k joins in MapReduce [1].

Acknowledgments

The work of A. Vlachou was supported by the Action "Supporting Postdoctoral Researchers" of the Operational Program "Education and Lifelong Learning" (Action's Beneficiary: General Secretariat for Research and Technology), and is co-financed by the European Social Fund (ESF) and the Greek State. The research of C. Doulkeridis has been co-financed by ESF and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Aristeia II, Project: ROADRUNNER.

8. REFERENCES

[1] C. Doulkeridis and K. Nørvåg. A survey of analytical query processing in MapReduce. *VLDB J.*, 23(3):355–380, 2014.

[2] J. Finger and N. Polyzotis. Robust and efficient algorithms for rank join evaluation. In *Proc. of SIGMOD*, pages 415–428, 2009.

[3] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting top-k join queries in relational databases. *VLDB J.*, 13(3):207–221, 2004.

[4] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-*k* query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):11:1–11:58, 2008.

[5] A. G. Parameswaran and H. Garcia-Molina. Recommendations with prerequisites. In *Proc. of RecSys*, pages 353–356, 2009.

[6] S. B. Roy, S. Amer-Yahia, A. Chawla, G. Das, and C. Yu. Constructing and exploring composite items. In *Proc. of SIGMOD*, pages 843–854, 2010.

[7] K. Schnaitter and N. Polyzotis. Optimal algorithms for evaluating rank joins in database systems. *ACM Trans. Database Syst.*, 35(1), 2010.

[8] M. Xie, L. V. S. Lakshmanan, and P. T. Wood. Breaking out of the box of recommendations: from items to packages. In *Proc. of RecSys*, pages 151–158, 2010.