Noname manuscript No.
(will be inserted by the editor)

Rank-Join Indices

Themis Palpanas (Paris Descartes Univ.), Panayiotis Tsaparas (Univ. of Ioannina)

Synonyms

Skyline queries, Top-k queries

Definition

Let R_1 be a relation, over attributes $A_1^1, \ldots, A_n^1$. We say that R_1 is a *ranked relation*, if there is a designated *rank attribute* A_r^1, with domain a subset of $\mathbb{R}^+$, such that the value $A_r^1(t)$ for tuple t defines the score of the tuple, and induces a ranking for the tuples in R_1. Let $R_1, \ldots, R_q$ be ranked relations. Without loss of generality, assume that $A_1^1, \ldots, A_1^q$ are the rank attributes, let θ be an arbitrary join condition defined between (sub)sets of the remaining attributes, and let $J(R_1, \ldots, R_q) = (R_1 \bowtie_{\theta_1} \ldots \bowtie_{\theta_{q-1}} R_q)$ denote the resulting relation. Let $f : \mathbb{R}^+ \times \ldots \times \mathbb{R}^+ \to \mathbb{R}^+$ be a scoring function that takes as input the rank attribute values $(s_1, \ldots, s_q) = (A_1^1(t), \ldots, A_1^q(t))$ of tuple $t \in J(R_1, \ldots, R_q)$, and produces a score value $f(s_1, \ldots, s_q)$ for the tuple t. A *top-k join query* asks for the k tuples from relation $J(R_1, \ldots, R_q)$ with the highest score f. A Rank-Join Index (RJI) is an index defined over the relation $J(R_1, \ldots, R_q)$, such that given a class of scoring functions $\mathcal{L}$ and an integer K, we can provide answers with guaranteed performance on any *top-k* join query, for any scoring function $f \in \mathcal{L}$, and $k \leq K$.

Historical Background

A plethora of data sources contain data entities that could be ordered according to a variety of attributes associated with the entities. Such orderings result effectively in a ranking of the entities according to the values in the attribute domain. The problem of aggregating such ranked relations has received considerable attention [8,7,6].

Most of the work in the literature assumes that there exists a fixed scoring function f for scoring the aggregate relation, and the goal is to efficiently compute the top-k ranked join query results for two relations R and S and a join predicate that are specified at query time. Agrawal and Wimmers [1] proposed a framework for preference based query processing. Various works considered realizations of a specific instance of this framework, namely *top-k* selection queries, that is, quickly identifying k tuples that optimize scores assigned by monotone linear scoring functions on a variety of ranked attributes and user specified preferences [9,5,10,2,4]. Most of these techniques for answering *top-k* selection queries [9,10,5,2] are not based on indexing. Instead, they are geared towards optimizing the number of tuples examined in order to identify the answer under various cost models of interest. Such optimizations include, minimization of tuples read sequentially from the input [10,9,5], or minimization of random disk access [2,3]. Chang et al. [4] propose an indexing technique for answering *top-k* selection queries. This technique does not provide guarantees for its performance and in the worst case, the entire data set has to be examined in order to identify the correct answer to a *top-k* selection query. Natsev et al. [12] proposed techniques to answer *top-k* queries over the join of two relations. They assume no preprocessing and compute the join of the relations from scratch for each join condition and user supplied preference values, thus being able to support ad hoc join conditions, but cannot provide performance guarantees for general data distributions and arbitrary user preferences. An efficient implementation of a pipelined operator for ranked joins was described in [11].

Address(es) of author(s) should be given

A ranked join index assumes fixed relations and a fixed join predicate, and builds an index that allows efficient retrieval of the top-k rank-join query results for a scoring function that is specified at query time. This idea was introduced in [14], where the geometric representation and mathematical underpinnings for the problem were also discussed in detail. This idea was later extended using the notion of convex layers [13], leading to improvements in the time performance for creating the RJI.

Scientific Fundamentals

The approach presented in [14] constitutes the *first* solution providing performance guarantees for *top-k* join queries over *two* relations (when preprocessing to construct a ranked join index for a specific join condition is permitted).

Preliminaries: Given two relationships R_1 and R_2, and a predicate θ we use $J(R_1, R_2)$ to denote the join relationship. Each tuple $t \in J(R_1, R_2)$ is associated with a pair of scores (s_1, s_2), corresponding to the scores for the rank attributes for relations R_1 and R_2 respectively. The scoring function $f : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$ produces a score $f(s_1, s_2)$ for the tuple t. Abusing the notation we will often use $t = (s_1, s_2)$ to denote a tuple of the join relation, and $f(t)$ to denote the score of the tuple.

The goal of the Rank-Join index is to build a structure over the tuples $t = (s_1, s_2)$ in $J(R_1, R_2)$ so as to efficiently answer top-k join queries for any function f. Let D denote the domain of the join relation, i.e., the set of all tuples. A key observation that is detailed in [14] is that, given an upper bound K for the value of k, we do not need to index the full set of tuples D. Some tuples are *dominated* by at least K tuples, so they can never be part of the top-k query result. Instead, we only need to index the set of dominating tuples D_K. The computation of D_K is beyond the scope of this exposition, and we refer the reader to [14] for more details. For the following we assume that we are given the set D_K as input.

Scoring Functions: An RJI is defined with respect to a class of functions $\mathcal{L}$ for which it can answer top-k queries. The indexes under consideration are with respect to the class of *Linear Monotone Functions.*

Definition 1 (Monotone Functions) *A function* $f : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$ *is* monotone *if the following holds: if* $x_1 \leq x_2$*, and* $y_1 \leq y_2$*, then* $f(x_1, y_1) \leq f(x_2, y_2)$*.*

Definition 2 (Linear Functions) *A linear scoring function* $f : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$ *is defined as* $f(x_1, x_2) = p_1x_1 + p_2x_2$*, where* $p_1, p_2 \in R^+$ *are the preference values of the scoring function. We use* $\mathcal{L}$ *to denote the set of all linear functions.*

Geometric Representation: A linear monotone scoring function $f \in \mathcal{L}$ is completely defined by a pair of preference values (p_1, p_2). Similarly a tuple in D_K is completely defined by the pair of scores (s_1, s_2). The key observation is that the function f can represented by the vector $\mathbf{e} = \langle p_1, p_2 \rangle$ on the plane. We use $f_\mathbf{e}$ to denote the scoring function defined by the vector $\mathbf{e}$. Similarly, a tuple $t \in D_K$ is represented by a vector of score values $\mathbf{s} = \langle s_1, s_2 \rangle$. In light of this geometric representation the value of a function $f_\mathbf{e}$ on a tuple $t \in \mathcal{D}_K$ with score vector $\mathbf{s}$ is the inner product of the vectors e, and s, that is $f_\mathbf{e}(\mathbf{s}) = p_1s_1 + p_2s_2$.

Without loss of generality we assume that $\|\mathbf{e}\| = 1$, that is, $\mathbf{e}$ is a unit vector. Note that if $\|\mathbf{e}\| = \alpha$ all scores are scaled by a factor of α and thus there is no change in the results of a top-k query. In this case the value of the scoring function $f_\mathbf{e}(\mathbf{s})$ is the projection length of the vector $\mathbf{s}$ on $\mathbf{e}$. An example is shown in Figure 1(a). For a set of ℓ tuples $\{t_1, t_2, ..., t_\ell\}$, the ordering defined by the scoring function $f_\mathbf{e}$ is naturally defined by projecting the points onto vector $\mathbf{e}$ and then ordering them from right to left. Figure 1(b) presents an example of such an ordering. We use $Ord_\mathbf{e}(\{t_1, t_2, ..., t_\ell\})$ to define this ordering, and $\overline{Ord_\mathbf{e}}(\{t_1, t_2, ..., t_\ell\})$ to denote the *reverse* of that ordering. Also, let $T_k(\mathbf{e})$ denote the result of the top-k query for the function $f_\mathbf{e}$. The set $T_k(\mathbf{e})$ consists of the first k tuples $Ord_\mathbf{e}(\{t_1, t_2, ..., t_\ell\})$.

In order to build the rank-join index we need to understand how the ordering $Ord_\mathbf{e}(\mathcal{D}_K)$ of the tuples change depending on the choice of the vector $\mathbf{e}$. Note that a two-dimensional unit vector $\mathbf{e}$, is fully defined by the *angle* $a(\mathbf{e})$ of the vector to the x-axis as shown in Figure 1(b). Note that since we assume that the preference values are positive, the angle $a(\mathbf{e})$ takes values in the interval $[0, \frac{\pi}{2}]$.

Consider now two tuples $t^1, t^2 \in D_K$. We will study their relative order when we change the value of $a(\mathbf{e})$. Let $\mathbf{s}^1 = (s^1_1, s^1_2)$, and $\mathbf{s}^2 = (s^2_1, s^2_2)$ be score vectors for the two tuples. Let $\langle \mathbf{s}^1\mathbf{s}^2 \rangle = \mathbf{s}^2 - \mathbf{s}^1$ denote the vector defined by the difference of $\mathbf{s}^2$ and $\mathbf{s}^1$, and let $b(\langle \mathbf{s}^1\mathbf{s}^2 \rangle)$ denote the angle of the vector $\langle \mathbf{s}^1\mathbf{s}^2 \rangle$ with the x-axis. Figure 1(c) presents an example. The ordering of tuples t_1 and t_2 for the different choices of $\mathbf{e}$ is governed by the following lemma.

Lemma 1 *Let* $t_1, t_2 \in \mathcal{D}_K$*, with score vectors* $\mathbf{s}^1$ *and* $\mathbf{s}^2$*, and let* $\mathbf{e}$ *be a preference vector. If* $b(\langle \mathbf{s}^1\mathbf{s}^2 \rangle) \in [0, \frac{\pi}{2}]$ *then* $Ord_e(\{t_1, t_2\})$ *is the same for any choice of the preference vector* $\mathbf{e}$*. Otherwise, let* $\mathbf{e}_s$ *be the vector orthogonal to* $\langle \mathbf{s}^1\mathbf{s}^2 \rangle$*. We have:*

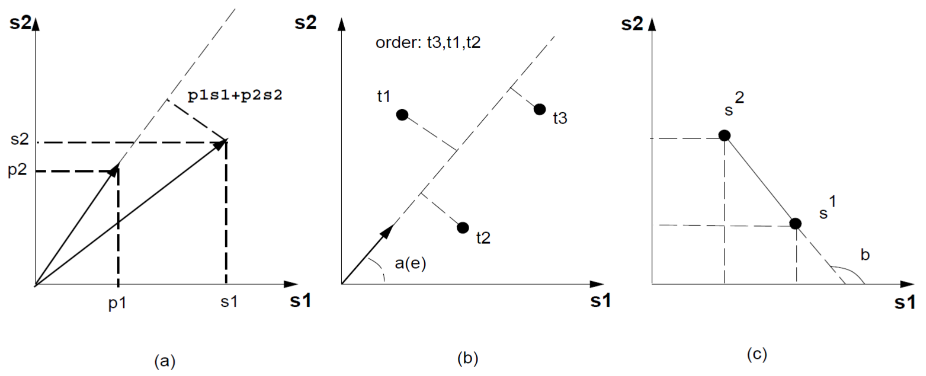

Fig. 1 Illustration of the Vector representation of scoring functions and rank attribute values.

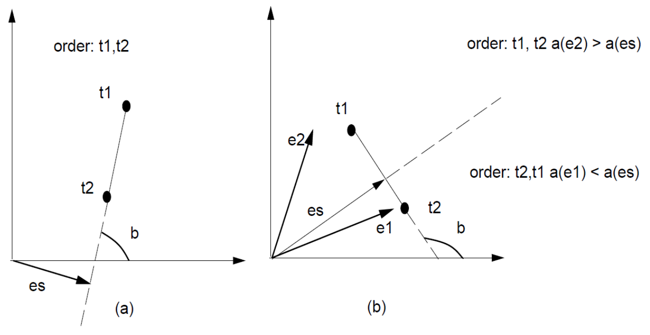

Fig. 2 Illustration of the two cases of Lemma 1.

1. $f_{\mathbf{e}_s}(\mathbf{s}^1) = f_{\mathbf{e}_s}(\mathbf{s}^2)$,
2. $Ord_{\mathbf{e}_1}(\{t_1, t_2\}) = Ord_{\mathbf{e}_2}(\{t_1, t_2\})$, *for all vectors* $\mathbf{e}_1, \mathbf{e}_2$ *with* $a(\mathbf{e}_1), a(\mathbf{e}_2) > a(\mathbf{e}_s)$, *or* $a(\mathbf{e}_1), a(\mathbf{e}_2) < a(\mathbf{e}_s)$,
3. $Ord_{\mathbf{e}_1}(\{t_1, t_2\}) = \overline{Ord_{\mathbf{e}_2}}(\{t_1, t_2\})$, *for all* $\mathbf{e}_1, \mathbf{e}_2$, *such that* $a(\mathbf{e}_1) < a(\mathbf{e}_s) < a(\mathbf{e}_2)$.

Figure 2 shows two examples for Lemma 1. Figure 2(a) shows a case where $b(\langle \mathbf{s}^1\mathbf{s}^2 \rangle) \in [0, \frac{\pi}{2}]$. The ordering of tuples t_1 and t_2 remains the same for any choice of preference vector $\mathbf{e}$. On the other hand, in Figure 2(b) we see that the vector $\mathbf{e}_s$ separates the positive quadrant into two areas. For preference vectors above $\mathbf{e}_s$, t_1 is ordered before t_2, while in the area below $\mathbf{e}_s$ t_2 is ordered before t_1. We call $\mathbf{e}_s$ the *separating vector* of tuples t_1, t_2, also denoted as $\mathbf{e}_s(t_1, t_2)$. We call the angle $a(\mathbf{e}_s)$ the *separating point*. The case were there are more than two tuples that share the same separating vector is discussed in [14].

Constructing a Rank Join Index: We now describe the construction of an RJI. First, for every pair of tuples $t_i, t_j \in \mathcal{D}_K$ we compute the separating vector $\mathbf{e}_s(t_i, t_j)$ and the corresponding separating point $a(\mathbf{e}_s)$. We then let a vector $\mathbf{e}$ sweep the positive quadrant of the plane, from the x axis to the y-axis, and we keep track of the composition of $T_K(\mathbf{e})$. Every time vector $\mathbf{e}$ crosses a separating vector $\mathbf{e}_s$, $Ord_{\mathbf{e}}(\mathcal{D}_K)$ changes by swapping two (or more if they are co-linear) *adjacent* tuples. If the swap causes the composition of $T_K(\mathbf{e})$ to change, then we store the new set $T_K(\mathbf{e}_s)$ as well as the separating vector $\mathbf{e}_s$ and the corresponding separating point $a(\mathbf{e}_s)$. At the end of this process, the stored separating vectors define a partition of the positive quadrant into regions, where each region has a different set $T_K(\mathbf{e}_s)$. Each separating vector $\mathbf{e}_s$ is defined by the separating point $a(\mathbf{e}_s)$. The separating points define a partition of the interval $[0, \pi/2]$. We index this partition into a B-tree, where each leaf node of the B-tree corresponds to a region of the positive quadrant, and a different set $T_K(\mathbf{e}_s)$. At query time, given a preference vector $\mathbf{e}$ we compute the angle $a(\mathbf{e})$ and use it to query the B-tree. The resulting set $T_K(\mathbf{e})$ is the answer to the top-k rank-join query. For additional details we refer the reader to [14].

Critical to the size of the index is M, the number of separating vectors identified by the algorithm. We provide a worst case bound on M by bounding the number of times that a tuple can move from position $K+1$ to position K in $Ord_{\mathbf{e}}(\mathcal{D}_K)$. Since every separating vector causes the swap of two *adjacent* elements in $Ord_{\mathbf{e}}(\mathcal{D}_K)$, the separating vectors that we index are the ones that cause a swap of the elements in positions K and $K+1$ in $Ord_e(\mathcal{D}_K)$, since these are the ones that cause the composition of $T_K(\mathbf{e})$ to change. For some tuple $t \in \mathcal{D}_K$ let $rank_{a(\mathbf{e})}(t)$ denote the position of tuple t in the ordering $Ord_{\mathbf{e}}(\mathcal{D}_K)$. The following lemma provides the means for bounding the value of M.

Lemma 2 *For a tuple* $t \in \mathcal{D}_K$, $rank_{a(\mathbf{e})}(t)$ *can change from* $K+1$ *to* K *at most* K *times as* $a(\mathbf{e})$ *ranges from 0 to* $\pi/2$.

The following theorem bounds the time and space complexity of the construction and query time of an RJI. The space-time trade-offs for RJI are studied in [14].

Theorem 1 *Given a set of dominating points* $\mathcal{D}_K$, *we can construct an index for top-k join queries in time* $O(|\mathcal{D}_K|^2 \log |\mathcal{D}_K|)$ *using space* $O(|\mathcal{D}_K| K^2)$ *providing answers to top-k join queries in time* $O(\log |\mathcal{D}_K| + k \log k)$, $k \leq K$.

Key Applications

The RJI finds applications in every domain containing data entities that can be ranked, and where user preferences towards specific attributes can be incorporated within queries. Examples include information retrieval (with preferences on properties of the documents), e-commerce (with preferences on products, or sellers), and others.

Future Directions

Given the volatility and streaming nature of several data sources nowadays, the current RJI techniques should be extended in order to accommodate incremental operation. Another challenging research direction in this area is the development of a RJI suitable for high dimensionalities (i.e., number of input relations). In this context, techniques for RJI that provide approximate answers could also offer effective and efficient solutions.

Experimental Results

For the presented methods, there is an accompanying experimental evaluation in the corresponding references.

Cross References

Top-K Selection Queries on Multimedia Datasets

References

1. R. Agrawal and E. Wimmers. A Framework For Expressing and Combining Preferences. *Proceedings of ACM SIGMOD*, pages 297–306, June 2000.
2. N. Bruno, L. Gravano, and A. Marian. Evaluating Top-k Queries Over Web Accessible Databases. *Proceedings of ICDE*, April 2002.
3. Kevin Chang and Seung-Won Huang. Minimal Probing: Supporting Expensive Predicates for Top-k Queries. *Proceedings of ACM SIGMOD*, June 2002.
4. Yuan chi Chang, L. Bergman, V. Castelli, C.S. Li, M. L. Lo, and J. Smith. The Onion Technique: Indexing for Linear Optimization Queries. *Proceedings of ACM SIGMOD*, pages 391–402, June 2000.
5. D. Donjerkovic and R. Ramakrishnan. Probabilistic Optimization of Top-N Queries. *Proceedings of VLDB*, August 1999.
6. R. Fagin. Combining Fuzzy Information from Multiple Systems. *PODS*, pages 216–226, June 1996.
7. R. Fagin. Fuzzy Queries In Multimedia Database Systems. *PODS*, pages 1–10, June 1998.
8. R. Fagin and E. Wimmers. Incorporating User Preferences in Multimedia Queries. *ICDT*, pages 247–261, January 1997.
9. L. Gravano and S. Chaudhuri. Evaluating Top-k Selection Queries. *Proceedings of VLDB*, August 1999.
10. V. Hristidis, N. Koudas, and Y. Papakonstantinou. Efficient Execution of Multiparametric Ranked Queries. *Proceedings of SIGMOD*, June 2001.
11. Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Joining Ranked Inputs in Practice. pages 950–961, Hong Kong, China, August 2003.
12. A. Natsev, YuanChi Chang, J. Smith, Chung-Sheng Li, and J. S. Vitter. Supporting Incremental Join Queries on Ranked Inputs. *Proceedings of VLDB*, August 2001.
13. Sanjaya Singh. Ranked Selection Indexes for Linear Preference Queries. *MSc Thesis, Wichita State University, KS, USA*, 2011.
14. Panayiotis Tsaparas, Themis Palpanas, Yannis Kotidis, Nick Koudas, and Divesh Srivastava. Ranked join indices. In *ICDE*, pages 277–288, 2003.