PREFERENCE TOP-K QUERY

YU FAN
CAO JUNHONG
1. INTRODUCTION

• Top-k query is important for ranking query answers
• Top-k query is widely used in daily life, e.g. university ranking
• Reverse top-k query: find preferences q belongs to Q where an object O becomes top-k query with regard to q
1. INTRODUCTION

- A common way to rank answers is using scoring functions (preferences)
- Here the discussed preference is additive, giving weights on attributes
- The score of tuple $t = (x_1, ..., x_d)$ under preference $q: (y_1, ..., y_d) \rightarrow \sum w_i y_i$ is $\sum w_i x_i$
I. INTRODUCTION

- Naïve way is computational expensive and hard to maintain
- Interested in a few tuples, but need to calculate and sort all tuples.
- Static ranking, Dynamic ranking
2. PRACTICAL SOLUTIONS

- A) Onion (Convex Hull Layer)
- B) Threshold
- C) LPTA
A) ONION

Based on the theorem:

- Given a set of records R mapped to a d-dimensional space, and a linear maximization (or minimization) criterion, the max (or min) value is achieved at one or more vertices of the convex hull of R.
A) BASIC TECHNIQUE

- An iteration of convex hull partitioning until the last subset cannot be partitioned.
- Name each subset a layer and number it from outside
A) CONVEX HULL CONSTRUCTION ALGORITHM

Input: dataset R of data records
K=1;

while(sizeof(R)>0){

Construct convex hull of R and store records in set V;
Assign V to layer k;
R = R – V, k = k+1;
}


A) STRUCTURE

- Geometrically the structure looks like an onion:
A) PROCEDURE TO FIND TOP-K

1. return top-1 result by searching the points in outmost layer.
2. find the next result (top-2) by searching the remaining points in outmost layer and points in the next layer.
3. continues until top-k points are found.
A) PROCEDURE TO FIND TOP-K

• Process:
A) PERFORMANCE

- Small storage overhead:
- The overhead is the beginning and ending page location for each layer
- High computation time:
- The cost for constructing convex hulls are $O(N^{d/2})$
- Disk IO cost:
  - Bounded by $\sum_{k=1}^{N} |C_k|/B$, $B$ is number of records per page
A) MAINTENANCE

- Computational expensive
- Inserting a new point $p$:
  - Locate the layer $p$ belongs to, and then re-construct the layer and the inner layers iteratively
B) THRESHOLD

- The Threshold Algorithm (TA) scans multiple sorted lists, representing different rankings of the same set of objects. A threshold $T$ is maintained for the overall score of unseen objects.
B) ALGORITHM

1. Do sorted access in parallel to each of the \( m \) sorted lists \( L_i \). As an object \( R \) is seen under sorted access in some list, do random access to the other lists to find the grade \( x_i \) of object \( R \) in every list \( L_i \). Then compute the grade \( t(R) = t(x_1, \ldots, x_m) \) of object \( R \): If this grade is one of the \( k \) highest we have seen, then remember object \( R \) and its grade \( t(R) \).
B) ALGORITHM

2. For each list $L_i$, let $x_i$ be the grade of the last object seen under sorted access. Define the threshold value $t$ to be $\max(x_1, \ldots, x_m)$. As soon as at least $k$ objects have been seen whose grade is at least equal to $t$, then halt.

3. Let $Y$ be a set containing the $k$ objects that have been seen with the highest grades. The output is $\{(R, t(R)) \mid R \in Y\}$.
B) PROOF

- Assuming a object $z$ is not seen by TA, we have:
  
  $$t(z) = t(x_1, \ldots, x_m) \leq t(x_1, \ldots, x_m) = \tau.$$  

- Therefore the objects seen by TA contains the top-k results.
B) PERFORMANCE

- Bounded buffer:
- Only need to keep top-k results and related scores, and pointers to last objects seen in each sorted list
- Relatively high computational cost:
- $O(m+m(m-1)\times Cr/Cs)$
- Each sorted access requires $(m-1)$ random access
C) LINEAR PROGRAMMING TA

- A view is a sorted list w.r.t a predefined preference query. The size can vary, usually from query length $K$ to number of tuples $N$.
- For $i$-th attribute, set weight $w_i = 1$, and all other weights zero. Set the size of the view to be $N$. This view becomes $i$-th base view. TA algorithms utilizes these views.
C) LINEAR PROGRAMMING TA

- Similar to TA, but not necessarily using base views. The threshold is calculated by LP.
- For a view $V_1$ with preference function $f_1$, if the last checked tuple is $t_1$, then for all tuples $t'$ ranked after $t_1$, $f_1(t') \leq f_1(t_1)$. This is a linear restriction on attributes of $t'$.
- Any unseen tuples satisfy all restrictions for all views used. Linear programming gives the max possible score of unseen tuples under query preference.
- When this score is smaller than the current $k$-th best value among scanned tuples under query preference, the algorithm terminates, since no unseen tuples can be top-$k$. 
C) PERFORMANCE

• Time cost is heavily on random access i/o. If $r$ views are used, and $d$ tuples were fetched before the algorithms terminates, the i/o cost is $O(rd)$

• Using base views, LPTA becomes TA.

• Performance of LPTA, or the tuples to fetch, depend on the views used.

• In general, views that are similar to query, or close in terms of vector, provide better performance.
3. THEORETICAL RESULTS

- A) geometry basics
- B) halfspace range reporting
- C) top-k query to halfspace report
- D) reverse top-k query to halfspace report
A) GEOMETRY BASICS

- Dual of a point \( p = (p_1, ..., p_d) \) is hyperplane \( p^*: x_d = p_1 x_1 + \cdots + p_{d-1} x_{d-1} - p_d \)
- Dual of hyperplane \( h^*: x_d = a_1 x_1 + \cdots + a_{d-1} x_{d-1} + a_d \) is point \( p = (a_1, ..., a_{d-1}, -a_d) \)
- If \( p \) lies above \( h \), \( h^* \) lies above \( p^* \).
- Set of hyperplanes vertical to a unit vector \( w \) is dual to vertical line \( w^* \) (upward).
- Tuples \( O = \{ o_i \} \) as points, preferences \( Q = \{ q_i \} \) as hyperplanes vertical to preference vector.
- If \( o = \pi_k (q, O) \), \( o^* \) is the \( k \)th hyperplane in \( O^* \) intersected by \( q^* \).
- \( \pi_{\leq k} (q, O) \), the top-\( k \) tuples in \( O \) under preference \( q \), is dual to first \( k \) hyperplanes in \( O^* \) intersected by \( q^* \).
A) GEOMETRY BASICS

- For set of hyperplanes $H$ in $\mathbb{R}^d$, arrangement of $H, A(H)$, is partition of $\mathbb{R}^d$ into faces of equal or lower dimension, induced by $H$. When $d = 2$, this is arrangement of lines.

- Level of a point $p$ w.r.t $H$ is $|H_{\leq p}|$, where $H_{\leq p} = \{ h \in H | h \text{ lying at or below } p \}$

- For two points $p, q$ not in any hyperplane in $H$, they are in the same face iff for each hyperplane $h \in H, p, q$ are on the same side of $h$. Furthermore, they have the same level.

- Boundaries and intersections of hyperplanes naturally induce faces of lower dimensions.

- $A_k(H)$ is the closure of level-$k$ faces of $A(H)$. 
B) HALFSPACE RANGE REPORTING

- With preprocessed point set $P$ of $n$ points in $R^d$, for any query halfspace $\gamma$, report all points in $P \cap \gamma$
- The time cost is at least output size $k$
- Introduce a data structure which partition the space in a way that:
  - When the points to be reported for the query hyperplane are many, scan them directly. The time cost should be no more than output size.
  - When the points to be reported are few, recursively query in children nodes.
- Such partition exists.
B) HALFSPACE RANGE REPORTING

• A simplicial partition for $P$ is $\Pi = \{(P_1, \Delta_1), \ldots, (P_m, \Delta_m)\}$, where $P_i$ are pairwise disjoint subsets partitioning $P$, and each $\Delta_i$ is an open simplex containing $P_i$. Call $P_i$ classes of $\Pi$.

• A hyperplane $h$ crosses simplex $\Delta$ if $h \cap \Delta \neq \emptyset$ and $\Delta \not\subseteq h$

• Crossing number of $h$ relative to $\Pi$ is the number of $\Delta_i$ crossed by $h$

• $h$ is k-shallow relative to $P$ if one of the halfspaces induced by $h$ contains no more than $k$ points of $P$

• Then here is the partition theorem for shallow hyperplanes
B) HALFSPACE RANGE REPORTING

- $P$ is an $n$-point set in $\mathbb{R}^d (d \geq 2)$, $r$ is a parameter, $1 < r < n$. There exists a simplicial partition $\Pi$ for $P$, whose classes $P_i$ satisfy $\frac{n}{r} \leq |P_i| \leq 2\frac{n}{r}$, and such that the crossing number of any $\frac{n}{r}$-shallow hyperplane relative to $\Pi$ is $O\left(r^{\frac{1-2}{d}}\right)$ (for $d \geq 4$), or $O\left(\log r\right)$ (for $d = 2, 3$).

- Proof uses other theorems and lemmas in geometry and can be found in reference.
B) HALFSPACE RANGE REPORTING

- Then the data structure is as following:
  - Choose $0 < \beta < \frac{1}{d}$, for a node $v$ in the tree, set $n_v = |P_v|$, $r_v = n_v^{\beta}$. Choose a simplicial partition $\Pi_v$ of size $O(r_v)$ for $P_v$ with classes $\leq 2 \frac{n_v}{r_v}$, such that the crossing number of any shallow hyperplane will be at most $K_v = O(r_v^{1 - \frac{2}{d}})$. The $O(r_v)$ children of $v$ is sets in $\Pi_v$. Build an auxiliary data structure to report points of $P_v$ in a query halfspace in time $O(n_v^{1 - \frac{1}{d}} (\log n)^{O(1)} + k)$, with $k$ the output size.
  - This bounds comes from $T(n) \leq O\left(n^{1 - \frac{1}{d}}\right) + O\left(r^{1 - \frac{1}{d}}\right)T\left(\frac{2n}{r}\right), r = n^{1 - \frac{1}{d}}$, for simplex search.
B) HALFSPACE RANGE REPORTING

- Space of each level is $O(n)$, depth is $O(\log \log n)$, total space $O(n \log \log n)$.
- Query algorithm starts at root. At each node, recursively call following procedure: Detect all simplices of $\Pi_v$ crossed by the boundary hyperplane $h$ of query halfspace $\gamma$. If the number of such simplices $\geq K_v$, $h$ is not $\frac{n_v}{r_v}$-shallow, the auxiliary structure is used to report the points in $P_v \cap \gamma$. Otherwise, for simplices contained in $\gamma$, report all points in them, then proceed recursively to trees for simplices crossed by $h$. Stopping at leaves with a constant size.
- The total query time for a single halfspace is $O(n^{1-\frac{2}{d}}(\log n)^{O(1)} + k)$, $k$ is output size.
C) TOP-K QUERY TO HALFSPACE REPORT

- Report $\pi_{\leq k}(q, O)$, the top-k tuples in $O$ under preference $q$.
- Simply binary search the threshold.
- Time $O(\log n(q(n) + k))$
- By the structure of space partition tree mentioned before, running time can be improved to $O(q(n) + k)$
The question of reverse top-k query is:

Given set $O$ of tuples, set $Q$ of preferences, for a query tuple $o \in O$, find the subset of preferences $Q' = \{ q \in Q | o \in \pi_{\leq k}(q, O \cup \{o\}) \}$

The relation of dual space and reverse top-k query is given by the lemma:

For query tuple $o$, preference $q \in Q'$ iff $q^* \cap A_k(O^*)$ lies above hyperplane $o^*$

The algorithm uses all preferences and predefine tuples to calculate the k-level closure and all intersections $q^* \cap A_k(O^*)$. Then the algorithm tests new input tuple $o^*$ in $O(q(n) + t)$, $t$ as output size.
Figure: Level in dual space
REFERENCE