Making Interval-Based Clustering Rank-Aware

Julia Stoyanovich*
University of Pennsylvania
Philadelphia, PA, USA
jstoy@cis.upenn.edu

Sihem Amer-Yahia
Yahoo! Research
New York, NY, USA
sihem@yahoo-inc.com

Tova Milo†
Tel Aviv University
Tel Aviv, Israel
milo@cs.tau.ac.il

ABSTRACT

In online applications, such as online dating, users often query and rank large collections of structured items. Top results tend to be homogeneous, which hinders data exploration. For example, a dating website user who is looking for a partner between 20 and 40 years old, and who sorts the matches by income from higher to lower, will see a large number of matches in their late 30s who hold an MBA degree and work in the financial industry, before seeing any matches in different age groups and walks of life. An alternative to presenting results in a ranked list is to find clusters in the result space, identified by a combination of attributes that correlate with rank. Such clusters may describe matches between 35 and 40 with an MBA, matches between 25 and 30 who work in the software industry, etc., allowing for *data exploration of ranked results*.

We refer to the problem of finding such clusters as *rank-aware interval-based clustering* and argue that it is not addressed by standard clustering algorithms. We formally define the problem and, to solve it, propose a novel measure of locality, together with a family of clustering quality measures appropriate for this application scenario. These ingredients may be used by a variety of clustering algorithms, and we present BARAC, a particular subspace-clustering algorithm that enables rank-aware interval-based clustering in domains with heterogeneous attributes. We validate the effectiveness of our approach with a large-scale user study, and perform an extensive experimental evaluation of efficiency, demonstrating that our methods are practical on the large scale. Our evaluation is performed on large datasets from Yahoo! Personals, a leading online dating site, and on restaurant data from Yahoo! Local.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*; H.5.2 [**Information Systems**]: Information Interfaces and Presentation—*User Interfaces*

*This research was supported in part by NSF grant 0937060 to the Computing Research Association for the CIFellows Project.

†This research was supported in part by the Israel Science Foundation, by the US-Israel Binational Science Foundation, and by the EU grant MANCOOSI.

EDBT 2011, March 22–24, 2011, Uppsala, Sweden.

General Terms

Algorithms, Human Factors, Performance

Keywords

Data Exploration, Ranking, Clustering

1. INTRODUCTION

In online applications that involve large structured datasets, such as Yahoo! Personals and Trulia.com, there are often thousands of items that match a user's preferences. In dating sites, a user may specify the age, height, income, education, political affiliation, and religion of a potential match. In real estate applications, a user may describe his dream home by its location, size, and number of bedrooms. The number of matches is often very high, making *data exploration* an interesting challenge.

Typically users also specify ranking criteria for the retrieved items, e.g., a sort order on a single attribute, or a weighted combination of multiple attributes. For example, in a dating site, potential matches may be ranked by decreasing income or increasing age, while in a real estate site, available houses may be ranked by increasing price or decreasing size. Ranking helps users navigate the set of results by making sure that the items users see first are of high quality, according to their ranking criteria. However, ranking also brings the disadvantage of *match homogeneity*: users are often required to go through a large number of similar items before finding the next different item. This is illustrated in the following example.

EXAMPLE 1.1. Mike*, a user of an on-line dating site, is looking for a partner.* Mike *specifies that he is interested in women who are 20 to 40 years old, have at least a Bachelor's degree, and live in New York City, and requests that results be sorted on income from higher to lower. When inspecting the results,* Mike *notices that the top ranks are dominated by women in their late thirties with an MBA degree who work in finance, making $150K - $200K annually. It takes* Mike *a while to scroll down to the next set of matches that are different from the top-ranked ones. In doing so, he encounters women who are between 25 and 30 years old, have a BS degree, and work in the software industry making $75K-$125K. After additional data exploration,* Mike *realizes that there is a* correlation *between age, education level, and income. Having looked at the result set a bit longer,* Mike *finds a smaller group of women with age varying over the entire 20-40 range, who have a liberal arts education, live on the Upper East Side, and are independently wealthy (i.e., have a high income but do not work full-time).*

Mike *would have been able to identify the different high-income groups more easily if results were presented in* labeled clusters *such as "36-40 year-olds who have an MBA and work in finance", "25-30 year-olds who work in the software industry and make $75K-$125K", "matches with a liberal arts education who live on the*

Upper East Side", etc. With results presented in this way, Mike *would then be able to explore the result set, deciding to focus on one particular cluster, or to consider matches from several clusters.*

A key point that arises from this example is that a user who is browsing the result set sequentially, item by item, is only able to infer some trends and correlations in the data after seeing a significant number of items. Sequential presentation is not very helpful if the user is trying to explore the data, i.e., understand general properties of the dataset and identify groups of similar items, particularly if the dataset is large and some groups are much smaller than others. The complexity of manual data exploration further increases with more sophisticated ranking. So, *Mike*'s ranking function could compute a score that is inversely proportional to the distance between his geographic location and the location of his match, and directly proportional to the income of the match.

In this paper we build on the intuition of Example 1.1 and propose to discover *clusters* in the result space, identified by a combination of intervals that are *correlated with result quality*. Here, an interval is a combination of an attribute name and a range of values from that attribute's domain, e.g., $age \in [20, 24]$, or an attribute name and a single value if the domain of the attribute is unordered, e.g., $profession = \text{“}software''$. We refer to the problem of finding such clusters as the *rank-aware interval-based clustering problem*. The challenge is to find the most interesting interval combinations, and we take the point of view that an interesting set of intervals contains matches that are *best from among comparable* with respect to the user's ranking function. For example, the pair $age \in [25, 30]$, $profession = \text{“}software''$ identifies a rank-aware cluster if many of the top earners in the 25 to 30 age range coincide with the top earners who work in the software industry.

We now demonstrate the reasons why rank-unaware clustering is ineffective in our application scenario. Consider *James*, a user who shares *Mike*'s filtering conditions. Suppose that, unlike *Mike*, who ranks matches on decreasing income, *James* is interested in seeing the same matches sorted by age in increasing order. Which clusters are best for which user depends on the user's ranking preferences. In particular, *Mike* may appreciate seeing the matches who are between 25-30 years old and work in the software industry in the same cluster as 35-40 year-olds who work in retail, if the two groups of matches have similar income. On the other hand, *James* would prefer seeing the two groups in separate clusters, because they differ on age. However, a clustering method that does not account for ranking cannot distinguish between these two users, and is therefore inappropriate for rank-aware data exploration.

One may argue that treating the score, or rank, of each item as an additional attribute, and using it for clustering in a rank-unaware framework such as CLIQUE [2], is sufficient. However, as we now demonstrate, this approach fails to address rank-aware data exploration effectively, for two reasons. First, if income (or rank computed based on income) were used as a clustering dimension, in the same way as other attributes, then *Mike* may see a high number of clusters, not all of which are of potential interest to him, e.g., a cluster of 25-30 year-olds with a high income may appear alongside a cluster of 25-30 year-olds with a low income. If many clusters are discovered by the algorithm, the potentially more interesting ones may go unnoticed. Worse yet, the algorithm may decide to merge together clusters that are of high interest to *Mike* with those of low interest, resulting in a potentially large heterogeneous cluster with homogeneous results dominating the top ranks. We will demonstrate in Section 6 that this condition arises in real datasets.

Note that which clusters exist in the query result depends on the composition of the result set and on the ranking function, and both are specified by the user. For example, higher income and higher age often co-occur, but this correlation becomes less pronounced in retirement age. Furthermore, correlations between attributes may not hold for all of the user's matches, i.e., these correlations are *local*. Considering *Mike*'s matches (Example 1.1) in the Yahoo! Personals dataset, we observed that higher income correlates with higher age over the entire 20-40 age range, while the number of women with post-graduate education increases with age only until age 29, and then plateaus. Because rank-aware correlations are local, results of rank-aware clustering must be computed at query time, and they must account for locality.

Our solution is based on *subspace clustering* — a natural approach to use when items are described by a high number of attributes, and when correlations are local, i.e., they hold for particular combinations of attribute values. Subspace clustering is a feature selection technique that aims to uncover structure in high-dimensional datasets by looking for multiple, possibly overlapping, subsets of features, called *subspaces* [23]. Each subspace represents an alternative view of the data. Dimensionality of subspaces is usually lower than that of the original space, and so clusters identified in subspaces are more easily interpretable by the user.

Our approach is to make subspace clustering *rank-aware*, adapting it to our target application scenario. In the remainder of this paper we will present a *rank-aware interval-based clustering* framework. This framework identifies clusters of results that have three important properties — they have sufficient clustering quality, are tight, and are maximal. We now briefly describe these properties.

Clustering Quality The goal of rank-aware clustering is to find sets of intervals that *agree* with respect to the highest-ranking matches that they contain. We will formally define semantics of agreement at top-N by presenting a family of *rank-aware clustering quality measures* in Section 2.2.1. One measure, $\mathcal{Q}_{topN}$, treats the top-N items of each interval as sets, and is high when the intersection of these sets is large. Another measure, $\mathcal{Q}_{SCORE}$, accounts for the scores of the items in the intersection at top-N, and is high for clusters that contain many high-scoring items. Finally, $\mathcal{Q}_{SCORE\&RANK}$ accounts for both scores and ranks, and is high for clusters that contain many high-scoring items in high ranks.

Tightness An important question is one of defining the initial search space of the clustering algorithm, that is, of enumerating the set of intervals from which to build clusters. For attributes with ordered domains, two or more consecutive intervals over the same attribute may be concatenated to produce a larger interval. However, as we will show in Section 2.2.2, concatenating together consecutive intervals is only beneficial if neither of the intervals *dominates* the other with respect to the ranking function. Intuitively, one interval dominates another if the items in its top-N are strictly better, i.e., they have higher scores. For example, when ranking on income from higher to lower, we may observe that items in the top-N of $age \in [20, 24]$ have lower scores than those in the top-N of $age \in [25, 29]$. As a result, the top-N of the concatenated interval $age \in [20, 29]$ may coincide with the top-N of $age \in [25, 29]$. In this case concatenating together the intervals will add no new items to the top-N of $age \in [25, 29]$, and will eventually generate a misleading cluster description. This cluster will suffer from result homogeneity in top ranks and will not be *tight*.

Maximality A final requirement that we impose is that clusters must be *maximal*. Intuitively, we want to discover sets of intervals that correlate with result quality and that are as large as possible. For example, suppose we are given intervals $I_1 : age \in [20, 29]$, $I_2 : edu = MBA$, and $I_3 : income \in [75K, 100K]$. If two-dimensional clusters I_1I_2, I_1I_3, and I_2I_3 are discovered, as well as a three-dimensional cluster $I_1I_2I_3$, then only $I_1I_2I_3$ is presented

to the user, because it contains many of the same top-N items as its two-dimensional *subspaces*. We will formally define maximality in Section 2.2.3. Maximality is motivated by the requirements of a web-based application scenario, where only a limited number of clusters may be shown to a user at any one time.

This paper makes the following contributions: We formally define *rank-aware interval-based clustering*, and propose corresponding locality and clustering quality measures (Sec. 2). We demonstrate how rank-awareness may be introduced into the subspace clustering framework by presenting **BARAC**, bottom-up algorithm for rank-aware clustering, and analyze its complexity (Sec. 3). We experimentally validate the effectiveness of our approach with *a user study* involving more than 400 users of Yahoo! Personals (Sec. 4). We perform an extensive *evaluation of efficiency* of **BARAC** on Yahoo! Personals datasets (Sec. 5). We compare **BARAC** to density-based clustering on Yahoo! Local datasets (Sec. 6).

2. FORMALISM

In this section we formalize the *rank-aware interval-based clustering problem* for structured datasets.

2.1 Intervals and Subspaces

We now introduce the key notions of a *ranked interval* and a *subspace*, as well as their respective notions of *dominance* and *inclusion* that will be used throughout the paper.

We are given a dataset $\mathcal{D}$ where items are described by attribute-value pairs, including a special attribute id that uniquely identifies each item. Attributes belong to a set $\mathcal{A}$, and each attribute is represented by a name and a domain of values. We model three types of attributes: continuous (e.g., *age*), ordinal categorical (e.g., *education*), and categorical with no order on the values (e.g., *ethnicity*).

Our focus is on applications in which a user specifies a *filtering condition* over a subset of the attributes A of $\mathcal{A}$ that selects a subset of the items $D \subseteq \mathcal{D}$. The user also specifies a *ranking function* R that induces an order on the items in D by assigning a score $i.score$ to each item $i \in D$. Our approach treats the ranking function as a black box and is thus not restricted in the type of ranking functions it admits. We consider one user at a time, and omit D and R from our notation when these are clear from context.

DEFINITION 2.1 (RANKED INTERVAL). *A* ranked interval*, or simply* interval*, I is defined by a triplet $\langle a, low, high\rangle$, where $a \in \mathcal{A}$ is an attribute, and $low, high$ are values in the domain of a. I contains the items in D in which attribute a takes on a value in the $[low, high]$ range, and returns the items sorted by $i.score$. We use I^0 to designate an empty interval.*

This definition assumes that the attribute a is over an ordered domain and that $low \leq high$. For attributes with discrete unordered domains $low = high$, and an interval refers to a single value.

In our formalism we will often refer to the top-N items in I, which we denote by $top(I, N)$. We use standard competition ranking, e.g., "1224", and include *all items* that tie for the last rank into $top(I, N)$, thus sometimes returning more than N items.

We now define *interval dominance*, a rank-aware measure of locality that will be used below to define the search space of a clustering algorithm.

DEFINITION 2.2 (INTERVAL DOMINANCE). *Given two* consecutive *intervals I_1 and I_2 over attribute a, an integer N, and a threshold $\theta_{dom} \in (0.5, 1]$, we say that I_1* dominates *I_2 at top-N iff*

$$\frac{|top(I_1, N) \cap top(I_1 + I_2, N)|}{N} \geq \theta_{dom}$$

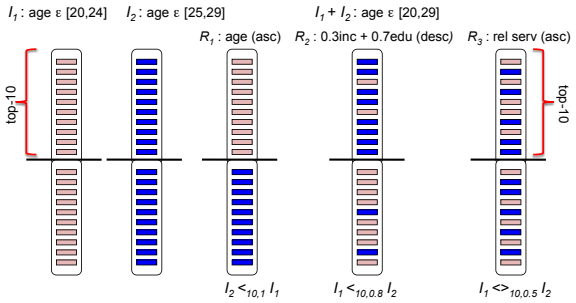


Figure 1: An illustration of interval dominance.

We denote this by $I_2 \prec_{N,\theta_{dom}} I_1$.

Here, $+$ is simply the concatenation of two consecutive intervals. For an attribute a with an ordered domain, I_1 and I_2 are consecutive if the values $I_1.high$ and $I_2.low$, or $I_2.high$ and $I_1.low$ are consecutive in the domain of a. No two intervals are consecutive if the domain of a is unordered. For any interval I, $I + I^0 = I$.

The intuition behind interval dominance is that the top-N items from the dominating interval are strictly better, w.r.t. the ranking function R, than the items in the the top-N of the dominated interval. Figure 1 illustrates dominance for intervals $I_1 : age \in [20, 24]$ and $I_2 : age \in [25, 29]$ for $N = 10$, and for three ranking functions. Consider the function R_1 that ranks results on age in ascending order, and observe that $top(I_1 + I_2, N) = top(I_1, N)$, and so $I_2 \prec_{10,1} I_1$. For the function $R_2 : 0.3inc + 0.7edu$ (descending), we observe that $\frac{8}{10}$ of $top(I_1 + I_2, N)$ originate from $top(I_2, N)$, and so $I_1 \prec_{10,0.8} I_2$. Finally, for R_3 that ranks matches on the frequency with which they attend religious services $top(I_1 + I_2, N)$ contains items from the top-10 lists of I_1 and I_2 in equal proportion, and so neither of the intervals dominates the other for any $\theta_{dom} \in (0.5, 1]$, which we denote by $I_1 \prec\succ_{10,0.5} I_2$.

So far in this section we focused on comparing, and concatenating together, *consecutive intervals over a single attribute*. We next define subspaces, in which *intervals over different attributes* are combined. A set of attributes defines a multi-dimensional space over which clustering operates, we will use the terms *attribute* and *dimension* interchangeably in the remainder of this section.

DEFINITION 2.3 (RANKED SUBSPACE). *A* ranked subspace*, or simply* subspace*, S is a set of intervals $\{I_1, \ldots, I_m\}$ over distinct attributes $a_1, \ldots, a_m$, interpreted as a conjunction of predicates specified by the intervals. A ranked subspace contains all items in D that satisfy all predicates, i.e., that belong to the intersection of the intervals. We refer to the number of intervals (and attributes) in a ranked subspace as its* dimensionality*. Items in S are ranked by the user's ranking function R.*

The following is a three-dimensional subspace: $S : \{age \in [20, 24], edu = MS, inc \in [75K, 100K]\}$. We overload notation and denote by $top(S, N)$ the top-N items that belong to S.

An important relationship that may hold between a pair of subspaces is *inclusion*, which we define below.

DEFINITION 2.4 (SUBSPACE INCLUSION). *Given two ranked subspaces S and S', we say that S' is* included *in S, denoted $S' < S$, if the set of intervals that define S' are a proper subset of the intervals that define S.*

For example, given $S' : \{I_1, I_2\}$ and $S : \{I_1, I_2, I_3\}$, we have that $S' < S$. Note that S' contains a *superset* of the items in S.

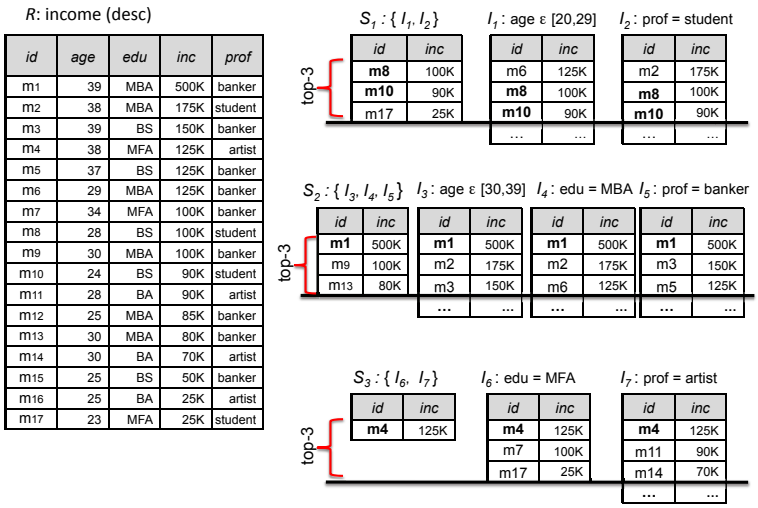


Figure 2: An illustration of clustering quality measures, with $N = 3$, $\theta_Q = \frac{2}{3}$, and ranking on income decreasing order.

2.2 Rank-Aware Clustering

As we argued in the introduction, we are interested in identifying subspaces that have three desired properties — rank-aware clustering quality, tightness, and maximality, and we refer to such subspaces as *clusters*. The three properties are explained below.

2.2.1 Rank-Aware Clustering Quality

The first property of a cluster is that it has sufficient clustering quality. In this work we consider three *rank-aware clustering quality measures*. All measures compare the quality of a subspace to a threshold $\theta_Q \in (0.5, 1]$. Thresholds are often used in clustering algorithms, and we will explore the effect of thresholds on performance in Section 5.

The first clustering quality measure is $\mathcal{Q}_{topN}$; it states that a k-dimensional subspace S has sufficient quality if $top(S, N)$ contains sufficiently many top-N items from each of its intervals.

$$\mathcal{Q}_{topN} : \frac{|top(S,N) \cap top(I_1,N) \cap \ldots \cap top(I_k,N)|}{N} \geq \theta_Q$$

We illustrate $\mathcal{Q}_{topN}$ using a fictional database of personal profiles in Figure 2, where we list the *id*, *age*, education (*edu*), income (*inc*), and profession (*prof*) of each profile. Items are sorted on income from higher to lower. Suppose that $N = 3$ and $\theta_Q = \frac{2}{3}$, i.e., that we are interested in agreement between intervals at top-3, and that intervals have to agree on at least 2 of the 3 items.

Consider intervals $I_1 : age \in [20, 29]$ and $I_2 : prof = student$, and note that matches m_8 and m_{10} are there in the top-3 lists of both intervals (ids designated in bold). The subspace $S_1 : \{I_1, I_2\}$ therefore has sufficient clustering quality according to $\mathcal{Q}_{topN}$. Consider now intervals $I_3 : age \in [30, 39]$, $I_4 : edu = MBA$, and $I_5 : prof = banker$. Only m_1 is common to the top-3 lists of all intervals, and so the subspace $S_2 = \{I_3, I_4, I_5\}$ does not have sufficient clustering quality according to $\mathcal{Q}_{topN}$.

The next measure, $\mathcal{Q}_{SCORE}$, states that a subspace has sufficient quality if it contains sufficiently many *high-scoring* items in its top-N. S will have the highest-scoring items in its top-N if these items are present in the top-N lists of all of its constituent intervals $I_1, \ldots, I_k$. In the best case, the top-N of the intersection of these subspaces will coincide with the top-N of their union, which gives rise to the formula:

$$\mathcal{Q}_{SCORE} : \frac{\Sigma_{i \in top(S,N)} i.score}{\Sigma_{i \in top(\cup_k I_k, N)} i.score} \geq \theta_Q$$

Consider again Figure 2, and note that, while S_2 only contains one top-3 item from the intersection of I_3, I_4, and I_5, namely, m_1, this item has a very high score. Items m_9 and m_{13} also have fairly high scores. We compute $\mathcal{Q}_{SCORE} = \frac{500+100+80}{500+175+150} = 0.824$, and so S_2 has sufficient clustering quality according to $\mathcal{Q}_{SCORE}$ with $N = 3$ and $\theta_Q = 0.8$. The final subspace in Figure 2, S_3, does not have sufficient quality for $\mathcal{Q}_{SCORE}$ for any $\theta_Q > 0.5$.

Finally, we present $\mathcal{Q}_{SCORE\&RANK}$ that models the relationship between item scores and ranks. The intuition is that a subspace with high-scoring items in high ranks may be just as interesting to the user as a subspace in which items have intermediate scores. We define this measure using NDCG (normalized discounted cumulated gain) [13], a common method for comparing ranked lists in Information Retrieval. NDCG quantifies the quality of a ranked list by comparing it to the best possible list, called the *ideal*. In our setting it is natural to use $top(\cup_k I_k, N)$ as the ideal.

$$\mathcal{Q}_{SCORE\&RANK} : AVG_{r \leq N} NDCG(top(S,N), top(\cup_k I_k, N))[r] \geq \theta_Q$$

We omit the details of NDCG computation (see [13] for more information), but note that subspace S_3 in Figure 2 has sufficient quality using $\mathcal{Q}_{SCORE\&RANK}$ with $\theta_Q = 0.65$.

Which clustering quality measure is appropriate depends on the distribution of scores in the result set, and we will discuss the interplay between the measures and the ranking function in Section 4.3.

An important property of clustering quality measures that allows for effective pruning of the search space is *downward closure*, defined below. Downward closure allows for early termination of *bottom-up subspace clustering algorithms* such as that presented in Section 3.

DEFINITION 2.5 (DOWNWARD CLOSURE). *We say that* downward closure *holds for a clustering quality measure $\mathcal{Q}$ iff, for any subspace S, if $\mathcal{Q}$ holds over S, then it also holds over every subspace S', s.t. $S' < S$ (as per Def. 2.4).*

We now show that downward closure holds for the three rank-aware clustering quality measures described above.

THEOREM 2.1. *The downward closure property holds for $\mathcal{Q}_{topN}$, $\mathcal{Q}_{SCORE}$, and $\mathcal{Q}_{SCORE\&RANK}$.*

PROOF. The fact that downward closure holds for $\mathcal{Q}_{topN}$ follows directly from the definition of $\mathcal{Q}_{topN}$ and from set properties, namely, that $|A \cap B| \leq min(|A|, |B|)$. For a one-dimensional subspace with N or more items, $\mathcal{Q}_{topN} = 1$. As subspace dimensionality increases, new sets are added to the intersection in the numerator of the expression. Thus the value of $\mathcal{Q}_{topN}$ is non-increasing with increasing dimensionality.

Downward closure holds for $\mathcal{Q}_{SCORE}$ and $\mathcal{Q}_{SCORE\&RANK}$. This is because the top-N of any subspace S consists of the items that are either in the top-N of all its constituent intervals ($i \in \cap_k top(I_k, N)$), or of items that have lower scores: $j \in (top(S, N) \setminus \cap_k top(I_k, N))$. The portion of $\cap_k top(I_k, N)$ in $top(S, N)$ is non-increasing as more intervals are added to the intersection. Thus, the value of the numerator of the $\mathcal{Q}_{SCORE}$ expression, and the DCG values in $\mathcal{Q}_{SCORE\&RANK}$ are strictly non-increasing in subspace dimensionality. At the same time, the denominator of $\mathcal{Q}_{SCORE}$, and the values of DCG_{Ideal} for $\mathcal{Q}_{SCORE\&RANK}$ are non-decreasing in the size of the union. Therefore, the values of $\mathcal{Q}_{SCORE}$ and $\mathcal{Q}_{SCORE\&RANK}$ are strictly non-increasing with increasing dimensionality. □

2.2.2 Tightness

The second desirable property of rank-aware clusters is tightness. We first define it formally, then illustrate it using examples.

DEFINITION 2.6 (TIGHT SUBSPACE). *For a given clustering quality measure $\mathcal{Q}$, an integer N and a threshold $\theta_{dom} \in (0.5, 1]$, we say that a k-dimensional subspace $S : \{I_1, \ldots, I_k\}$ is tight if there does not exist another k-dimensional subspace $S' \neq S$, $S' : \{I_1 + I'_1, \ldots, I_k + I'_k\}$ over the same attributes $a_1, \ldots, a_k$ where, for all j, $I'_j = I^0$ or $I_j \prec_{N,\mathcal{Q}_{dom}} I'_j$ or $I_j \succ_{N,\mathcal{Q}_{dom}} I'_j$ (as per Def. 2.2).*

Recall that, for numerical and ordinal categorical attributes, two or more consecutive intervals may be concatenated to produce a larger interval, which may or may not be beneficial. Consider again Figure 1 with intervals $I_1 : age \in [20, 24]$ and $I_2 : age \in [25, 29]$. For R_1 that ranks matched on increasing age, the top-10 items in $I_1 + I_2$ originate from I_1. More interestingly, for R_2 that ranks matches on a combination of income and education, 8 of the top-10 items in $I_1 + I_2$ originate from I_2, reflecting that there is a correlation between higher income and higher age in the 20 to 29 age range. In both cases, it is not beneficial to add $I_1 + I_2$ to the search space, because a cluster that incorporates $I_1 + I_2$ will contain the same, or nearly the same, items at the top-N as a cluster that incorporates just the dominant interval. In this sense, having a larger interval $I_1 + I_2$ in the cluster description adds no information. We refer to a cluster that incorporates the dominant interval (e.g., I_2 in the case of R_2) as *tight* according to Definition 2.6, while a cluster with $I_1 + I_2$ is not tight. A cluster that incorporates the dominated interval, e.g., I_2 for R_1, is also tight, since no other age interval has the same items at the top-N as I_2.

Consider now the ranking function R_3 in Figure 1. Here, neither of the intervals I_1 and I_2 dominates the other, i.e., the top-N of $I_1 + I_2$ differs considerably from the top-N of both I_1 and I_2. In this case adding I_1, I_2, and $I_1 + I_2$ to the search space is beneficial, because it may allow the algorithm to discover clusters that would not be identified by looking at either I_1 or I_2 individually. Clusters that incorporate $I_1 + I_2$ in this case are *tight*.

2.2.3 Maximality

The final desirable property of rank-aware clusters is maximality.

DEFINITION 2.7 (MAXIMAL SUBSPACE). *For a given clustering quality measure $\mathcal{Q}$, we say that a subspace S is maximal if it has sufficient quality according to $\mathcal{Q}$ and if there does not exist another subspace S' such that S' also has sufficient quality, and $S < S'$ (as per Def. 2.4).*

Going back to our example in Figure 2, S_2 is a maximal subspace, while $S'_2 : \{I_3, I_4\}$ is not.

2.3 Problem Statement

We are now ready to define *rank-aware clusters* and to formally state the problem addressed in this paper.

DEFINITION 2.8 (RANK-AWARE CLUSTER). *For a given clustering quality measure $\mathcal{Q}$, we say that rank-aware subspace S is a cluster if its clustering quality is higher than a given threshold θ_Q, it is maximal, and it is tight.*

The *rank-aware interval-based clustering problem* is stated as follows. For a dataset D, a scoring function R, a rank-aware clustering quality measure $\mathcal{Q}$, an integer N, a dominance threshold θ_{dom}, and a quality threshold θ_Q, find all clusters. We present an algorithm that solves this problem in the following section.

3. THE BARAC ALGORITHM

Our proposed algorithm **BARAC**, **B**ottom-up **A**lgorithm for **R**ank-**A**ware **C**lustering, has structure similar to CLIQUE [2], a now-classic bottom-up subspace clustering algorithm. A variety of subspace clustering algorithms exist, incorporating particular clustering quality measures, and navigating the search space top-down, bottom-up, or using a combined approach. We chose to build upon CLIQUE because its bottom-up search strategy is simple to implement and, as we will demonstrate in Section 5, is practical in our application domain. While our implementation is based on CLIQUE, we stress that the ingredients presented in Section 2 may be incorporated into other subspace clustering frameworks, such as those described in [23], making them rank-aware.

The pseudo-code for **BARAC** is presented in Algorithm 1. We describe the four subroutines of **BARAC** and analyze the complexity of the algorithm in the remainder of this section.

Algorithm 1 BARAC: **B**ottom-up **A**lgorithm for **R**ank-**A**ware **C**lustering

Require: dataset D, ranking function R, N, θ_Q, θ_{dom}, $maxClusters$
1: $grid = \mathbf{BuildGrid}(D, R, N)$;
2: $mergedGrid = \mathbf{Merge}(grid, D, N, \theta_{dom})$;
3: $subspaces = \mathbf{Join}(mergedGrid, \theta_Q)$;
4: $clusters = \mathbf{Choose}(clusters, maxClusters)$;
5: **return** $clusters$

3.1 BuildGrid

The procedure **BuildGrid** (Algorithm 2) starts by computing a score for each item $i \in D$, and then sorts the items in decreasing order of score. As the dataset is scanned, all distinct values for each attribute $a_i \in \mathcal{A}$ are recorded as $domain(a_i)$. Next, we consider each attribute a_i with a corresponding $domain(a_i)$, and compute a $grid$ data structure that is an array of one-dimensional histograms, one for each value in $domain(a_i)$. Each histogram bucket contains the list of top-N items, from among those that fall within the bucket. Bucket j for attribute i is denoted by $grid[i][j]$.

For attributes with high-cardinality domains, e.g., *height*, keeping a bucket per attribute value may be impractical, and we may instead split the domain into a bounded number of equi-width buckets, with the number of buckets specified by a parameter.

Algorithm 2 Procedure **BuildGrid**

Require: D, R, N
1: compute a score for each $i \in D$, sort items by $i.score$;
2: init $grid$, a matrix with one row per attribute $a_i \in \mathcal{A}$;
3: **for** $a_i \in \mathcal{A}$, where $|domain(a_i)| > 1$ **do**
4: **for** $val_j \in domain(a_i)$ **do**
5: {allocate 1 column in $grid[i]$ per value val_j}
6: $grid[i][j].range = [val_j, val_j]$;
7: $grid[i][j].items = top(\sigma_{a_i=val_j} D, N)$;
8: **end for**
9: **end for**
10: **return** $grid$

3.2 Merge

Merge runs multiple passes of the procedure **OnePassMerge** (Algorithm 3), building $mergedGrid$ — a dominance-based grid that contains all tight one-dimensional subspaces. **OnePassMerge** takes $grid$ as input, and expands the search space of the algorithm by considering, and possibly merging, runs of neighboring histogram buckets along the same dimension. For an attribute a_i, if neither of the two neighboring intervals $grid[i][j]$ and $grid[i][k]$ dominates the other (Definition 2.2), then the concatenation of the intervals is added to the grid.

When a run of **OnePassMerge** completes, it is invoked again on the output grid, and explores merging additional intervals. The process terminates when no new intervals are added.

Algorithm 3 Procedure **OnePassMerge**

Require: $grid, D, N, \theta_{dom}$
1: $mergedGrid = cloneGrid(grid)$;
2: **for** $a_i \in \mathcal{A}$ **do**
3: **for** $j = 1$ to $grid[i].length - 1$ **do**
4: {Check dominance among consecutive intervals.}
5: $k = j + 1$;
6: **if** $(grid[i][j] \prec\succ_{N,\theta dom} grid[i][k])$ **then**
7: {+ denotes interval concatenation.}
8: $grid[i][j+k].range = [grid[i][j].low, grid[i][k].high]$;
9: $grid[i][j+k].items = top(\sigma_{a_i \in grid[i][j+k].range} D, N)$;
10: $addToGrid(mergedGrid[i], grid[i][j+k]$;
11: **end if**
12: **end for**
13: **end for**
14: **return** $mergedGrid$

We make three observations about **Merge**. First, the output grid is typically much larger than the original grid. Second, the lower the threshold θ_{dom}, the fewer intervals are generated. We explore the impact of θ_{dom} on efficiency in Section 5.4. Third, all subspaces constructed from the intervals produced by **Merge** are guaranteed to be tight (as per Definition 2.6).

3.3 Join

The procedure **Join** computes high-quality subspaces in higher dimensions by progressively joining together lower-dimensional subspaces. Note that one-dimensional subspaces trivially have sufficient clustering quality, for any rank-aware clustering quality measure, and for any value of θ_Q. Hence, so do all the subspaces computed by **Join**. This procedure is similar to other bottom-up subspace clustering algorithms, and was first proposed in CLIQUE [2].

Algorithm 4 Procedure **Join**

Require: $mergedGrid, \theta_Q$
1: $Subspaces_{K-1} = mergedGrid$;
2: $result = Subspaces_{K-1}$;
3: **repeat**
4: $Subspaces_K = \mathbf{doJoin}(Subspaces_{K-1}, \theta_Q)$;
5: $result = result \cup Subspaces_K$;
6: **until** $Subpsaces_K = \emptyset$
7: **return** $result$;

Algorithm 5 Procedure **doJoin**

Require: $Subspaces_{K-1}, \theta_Q$
1: $Subspaces_K = \emptyset$;
2: **for** $S_1 \in Subspaces_{K-1}$ **do**
3: **for** $S_2 \in Subspaces_{K-1}$ **do**
4: **if** $compatible(S_1, S_2)$ **then**
5: $append(Subspaces_K, joinSubspaces(S_1, S_2))$;
6: **end if**
7: **end for**
8: **end for**
9: $Subspaces_K = prune(Subspaces_K, \theta_Q)$;
10: **return** $Subspaces_K$;

Join is presented in Algorithm 4; it repeatedly invokes the subroutine **doJoin** and terminates when no more subspaces are identified that have sufficient clustering quality. Procedure **doJoin**, presented in Algorithm 5, takes $(k-1)$-dimensional subspaces and a quality threshold θ_Q as input, and returns a set of k-dimensional subspaces that have quality higher than θ_Q. This is done by first identifying a candidate set of k-dimensional subspaces (lines 2-8), and then pruning the set by removing all subspaces that do not pass the quality threshold (line 9). We omit pseudo-code for some of the subroutines, but describe them verbally below.

Two $(k-1)$-dimensional subspaces S_1 and S_2 are said to be *compatible* if they contain $k-2$ equal intervals, and if the attribute name of$(k-1)^{st}$ interval of S_1 is *lexicographically lower* than the attribute name of the $(k-1)^{st}$ interval of S_2. **joinSubspaces** outputs a k-dimensional subspace described by the union of the intervals (and the intersection of the items) of S_1 and S_2.

We will discuss run-time performance of the algorithm in Section 3.5, but note here that the lower the value of θ_Q, the higher the number of subspaces generated by **Join**. We will explore the impact of threshold values on performance in Section 5.4.

3.4 Choose

The final step of **BARAC** is a call to the procedure **Choose**. This procedure identifies all maximal clusters and then optionally performs cluster selection. Cluster selection is necessary in an online data exploration scenario, if more clusters were discovered by the algorithm than can be shown to the user. The parameter $maxClusters$ determines the number of clusters to be selected, and a *more clusters* button is available, and gives access to additional clusters discovered by **BARAC**.

We omit pseudocode for **Choose** due to space considerations, and describe it verbally. As the first step, **Choose** identifies *maximal* subspaces from among those produced by **Join**. These are all the clusters identified by **BARAC**. If more than $maxClusters$ were identified, additional selection is needed.

Recall that a distinguishing feature of subspace clustering is the ability to present multiple alternative views of the data to the user. We ran a preliminary user study in which we interviewed six users about their data exploration experience with an early version of our prototype, with the goal of establishing a meaningful cluster selection heuristic. The majority of our users commented that they pre-

fer seeing clusters with a variety of descriptions, in terms of both attribute names and value ranges.

In light of this we decided to choose clusters so as to *maximize diversity of cluster descriptions*. Other heuristics, such as maximizing total item score, or minimizing cluster overlap, are also possible, and we leave an in-depth study of cluster selection to future work. Our selection algorithm uses K-means to identify $maxClusters$ meta-clusters with similar descriptions. Similarity between clusters accounts for both attribute names and for value ranges. Having identified $maxClusters$ meta-clusters, we select a representative that has the highest total score up to top-10.

3.5 Complexity Analysis

The run-time complexity of **BARAC** is polynomial in the size of the input $|D|$, with the exponent determined by the number of distinct attributes in the schema, i.e., by the dimensionality of the space. In the worst case, the algorithm considers each subset of attributes, and, for each such subset, considers all combinations of consecutive intervals over the attribute. The number of such intervals is bounded by n^2, where n is the cardinality of the attributes. The worst-case exponential dependency in the number of attributes is unavoidable, as we now demonstrate.

EXAMPLE 3.1. *Consider a dataset in which exactly one item exists for all combinations of attribute values. Therefore, the number of items in D is the product of domain cardinalities of all attributes, and therefore exponential in the number of attributes. Further, assume that a scoring function assigns an arbitrary distinct score to each item.*

Now, suppose that $N = 1$, i.e., that we are interested in finding clusters in which the intervals agree on the top-1 element (using, e.g., $\mathcal{Q}_{topN}$). Following Definition 2.8, each item will correspond to a cluster. Thus, the number of clusters is also exponential in the number of attributes.

Such exponential dependency is not unexpected, since even counting the number of distinct maximal frequent itemsets is #P-complete [28]. However, as we will show experimentally in Section 5, in practice the algorithm terminates much sooner, for two reasons. First, correlations are local and clusters commonly are of low dimensionality. Second, we can prove that, for a given pair of attributes a_i and a_j, each interval over a_i is compatible with at most one interval over a_j. This is because the quality threshold requires that intervals agree on more than 50% of their top-N items to be compatible. This implies that as soon as the compatibility between a_i and a_j is established, the algorithm can stop looking for other a_i' for a_j (resp., a_j' for a_i).

4. EVALUATION OF EFFECTIVENESS

4.1 The Experimental Dataset

Dataset. We evaluated the performance of **BARAC** on a dataset from Yahoo! Personals, a leading on-line dating service with millions of registered users. Users of the service create a *personal profile* in which they describe themselves using 30 structured attributes. Users also commonly store one or several *target profiles*, expressed in terms of the same structured attributes. When specifying that profile, users designate attributes as *required* and *desirable*. Required attributes are used as *filtering* conditions for exact matching against personal profiles, while desirable attributes are used for *ranking* exact matches.

In our experiments we focus on computing matches for male users, whom we call *seekers*, as there are at least one order of magnitude more males searching for females. We use 19 of the total 30 attributes, because there was no meaningful correlation between the ignored attributes (e.g., astrological sign) and other attributes, making them less suitable for clustering. Two of the 19 attributes, *has photo* and *gender*, have only two distinct values, and we use them for filtering, but not for clustering. So, there are 19 filtering attributes and 17 clustering attributes in our dataset. Therefore, for any given query, the number of clustering dimensions is at most 17.

Ranking. The ranking functions we consider use 6 attributes: *age*, *height*, *body type*, *education*, *income*, and *religious services* (the frequency with which the user attends religious services). We chose these attributes because they are either continuous or ordinal categorical, thus inducing a natural order on their values. Which of the 6 attributes are included in the scoring function depends on which attributes are marked as *desirable* in the target profile.

The first scoring function we used, *attribute-rank*, assigns equal weights to each ranking attribute, and computes the score of an item as the sum of distances between the item and the ideal item along each attribute dimension. Here, an ideal item has the best possible value for each ranking attribute from among items in the filtered dataset. Distances along each dimension are normalized by the difference between extreme values for the corresponding attribute found in the filtered dataset. Note that this function is *personalized* in two ways. First, the user specifies which attributes are included in the scoring function. Second, the value of each ranking attribute contributes to the score based on how it compares to the best and worst values for that attribute, from among items that pass the filtering conditions of the target profile.

The second function, *geo-rank*, scales the value returned by *attribute-rank* by the geographic distance between the seeker and his match: $geo\_rank = \frac{attribute\_rank}{1+(geo\_distance/100)}$.

We will discuss in detail in Section 4.3 that, because the clustering outcome depends on the combination of a user's filtering condition and the distribution of scores imposed by a particular scoring function, it is not always possible to find a meaningful clustering. The intuition is that rank-aware clustering does not apply if ranking does not discriminate well between high-quality and low-quality results, that is, if all, or most, of the items in the result set are tied for the same score. For example, selecting users with *income = 50K*, and then ranking on income, is not helpful, since all users will share the same score. Users whose scoring function assigned the top score to more than 30% of their profile matches were excluded from our evaluation.

4.2 User Study

We evaluated the effectiveness of **BARAC** by inviting a subset of registered Yahoo! Personals users to participate in a user study. The dataset is described in Section 4.1; we used the complete live dataset for the user study. As is done by the dating service itself, we use location-based filtering (e.g., *find matches who live within 250 miles of Austin, TX*) in addition to attribute-based filtering. Our implementation achieves sub-second response times for most searches.

User Study Design. Our user study ran for a period of five weeks. 454 users participated in our study and executed 861 searches. We implemented the *attribute-rank* ranking function (see Section 4.1).

Our users are accustomed to ranked lists, which differ from **BARAC** clusters in two ways. First, the set of results that appear in high ranks, and that the user sees first, may be different, i.e., content is different. Secondly, results are presented in labeled clusters, i.e., presentation is different. Our user study was designed so as to isolate the effects of these two aspects. Figure 3 summarizes the design of our study, in which we compare four treatments.

The *top list* treatment shows the top-100 matches in a traditional

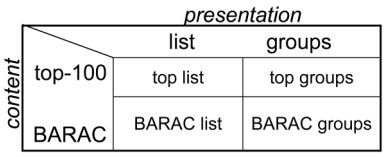


Figure 3: User study design matrix.

treatment	total searches	prod. searches
top list	331	17%
top groups	304	14%
BARAC list	100	15%
BARAC groups	126	20%

Table 1: Productive searches by treatment.

treatment	faves per search	faves per prod. search
top list	0.84	5.05
top groups	0.87	7.33
BARAC list	0.74	4.93
BARAC groups	1.55	12.38

Table 2: Number of faves per search.

ranked list. The *BARAC groups* treatment is our rank-aware clustering. Matches are clustered and 10 clusters are chosen and presented. The user may expand a cluster to see its content. All clustering is done with $\theta_Q = 0.7$, $\theta_{dom} = 0.51$. We will explore the effect of thresholds on quality in a future study. The *top groups* treatment isolates the effect of presentation, with data coming from the ranked list, i.e., the content is the same as in a ranked list. The user is presented with 10 groups of 10 results each. The first group corresponds to the first 10 matches and is labeled *Top 1-10*, followed by the next 10 matches labeled *Top 11-20*, etc. The *BARAC list* treatment presents matches produced by *BARAC groups* as a ranked list. The list is generated by taking the top-10 results from each group, then the next best results are added in a round-robin fashion from each group, until a total of 100 results are selected. The interface is the same as for *top list*.

Users are randomly assigned to one of the four treatments. However, as we discussed in Section 4.1, rank-aware clustering is not always applicable. In cases where *BARAC groups* or *BARAC list* do not apply due to scores being too uniform, the system defaults to *top groups* and *top list*, respectively.

We evaluate the effectiveness of treatments using an intuitive rating mechanism. Users are asked to *fave* an individual match or a group, by clicking on a star next to the match or group.

Results. The first metric we use to compare the treatments is the *percentage of productive searches* – searches that resulted in the user faving at least one match or group. Of 861 searches, 140, or 16%, were productive. Table 1 presents a break-down by treatment. We observe that *BARAC groups* has the highest percentage of productive searches, followed by *top list*. However, the difference between any two treatments is not statistically significant.

We next consider the effect of content and of presentation on the user experience. For *top groups* and *BARAC groups*, we compare the percentage of searches where users faved one or more groups, and the total number of faved groups. Of 304 total *top groups* searches, 12 resulted in faved groups. A total of 14 groups were faved in these 12 searches. The group *Top 1-10* was faved 11 times, and *Top 11-20* was faved 3 times. No other groups were faved, suggesting that user preference is guided by global ranking.

For *BARAC groups*, of 126 searches, 11 resulted in faved groups, and a total of 21 groups were faved. Groups with diverse descriptions were faved, indicating that preference is based on group descriptions and content rather than on global ranking. *BARAC groups* outperforms *top groups*, with a statistically significant difference in the number of searches with faved groups ($p < 0.05$).

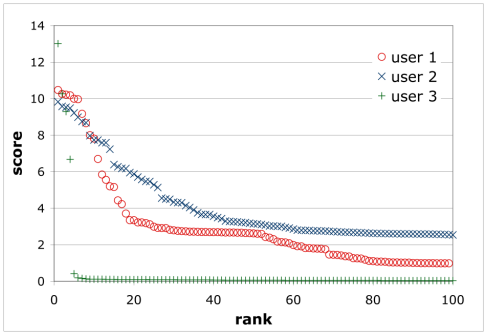


Figure 4: Top-100 for *geo-rank* for 3 users.

Finally, let us compare the four treatments according to a quality score that computes the average number of faves per search. We denote by F the number of distinct results that were faved by the user, either directly, or because the match was *in the top-10 of a faved group*. When a group is faved, all matches within the group are faved. Indeed, with the exception of one case, when users faved an entire group, they did not fave matches in that group, thereby implying that all matches in the group are faved. Note that top-10 lists of groups do not overlap in *top groups*, but may overlap in *BARAC groups*. We focus on the top-10 results for each group because this is the number of results that is accessible in *top groups*. More results may be accessed in *BARAC groups*, but we use 10 as the common denominator to make the methods comparable.

Table 2 presents the ratio of F to the total number of searches, and to the number of productive searches. We observe that *BARAC groups* significantly outperforms other methods, with an average of 1.55 faves per search, and 12.38 faves per productive search.

In summary, users are more engaged with *BARAC groups*, where they explore and fave more matches.

4.3 Choosing a Clustering Quality Metric

As we discussed in Section 2, $\mathcal{Q}_{topN}$ favors subspaces that contain many items that are in the top-N lists of their constituent intervals, irrespective of scores and ranks. Conversely, $\mathcal{Q}_{SCORE}$ and $\mathcal{Q}_{SCORE\&RANK}$ assign a higher score to a subspace in which the top-N lists of the constituent intervals intersect at top ranks, particularly if top-ranked items have significantly higher scores.

Ideally, a rank-aware clustering quality measure should be rich enough to capture the distribution of scores imposed by the ranking function. $\mathcal{Q}_{topN}$ treats all items with N highest scores equally, and is thus appropriate for ranking functions where the best N items have higher scores than the rest of the items, but where *there is no significant variability in scores among the top-N*. The ranking function *attribute-rank* is one such function (see Section 4.1). Conversely, $\mathcal{Q}_{SCORE}$ and $\mathcal{Q}_{SCORE\&RANK}$ are most meaningful if there is a significant variability in scores among items in the top-N. For example, for $N = 100$ it should hold that the first 10 items have much higher average scores than the following 10 items etc. The function *geo-rank* (see Section 4.1) is one such function.

However, while $\mathcal{Q}_{SCORE\&RANK}$ is sensitive to the distribution of scores, the same ranking function, e.g., *geo-rank*, may not generate a distribution of scores that is appropriate for $\mathcal{Q}_{SCORE\&RANK}$ for all users because their filtering conditions may differ. For users

who live in sparsely-populated areas this function may produce very few high-scoring items. Consider the distribution of top 100 scores for three users in Figure 4. Users $user_1$ and $user_2$ have similar distributions, while $user_3$ only has four high-scoring items in his top-100, followed by 96 items with similar low scores. $user_3$ is not a good candidate for $\mathcal{Q}_{SCORE\&RANK}$, and a score-insensitive quality measure like $\mathcal{Q}_{topN}$ should be used instead.

5. EVALUATION OF PERFORMANCE

Our experimental prototype is implemented in Java and operates on memory-resident data. All performance experiments in this section were executed on a 64-bit machine with two Intel Xeon 2.13GHz processors and 4GB of RAM, running RedHat EL AS 4. We focus on the $\mathcal{Q}_{topN}$ clustering quality measure and the *attribute-rank* scoring function in our performance experiments.

Performance experiments in this section were executed over a snapshot of the dataset that was taken during a recent consecutive one-month period, and contains all profiles that were registered with the dating service up to and including that month. Unlike in Section 4.2, where we used location-based filtering in conjunction with attribute-based, we do not use location-based filtering in the experiments in this section. This results in much larger datasets, and allows us to better test the scalability of **BARAC**.

We analyze performance in terms of three distinct stages: **BuildGrid**, **Merge**, and **Join**. See Section 3 for a description of these stages. We do not present performance numbers for the cluster selection stage, **Choose**, that uses the K-means algorithm to generate meta-clusters based on similarity of cluster description. K-means converged after at most 3 iterations in all cases, and the run-time overhead of this stage was under 10ms in all cases, or less than 0.1% of the total processing time.

User Sampling. We evaluated the performance for 100 target profiles. We chose a representative sample of profiles that cover a range of filtering and ranking attributes, as well as different number of matches. The chosen target profiles specify between 3 and 15 filtering attributes, and between 1 and 6 ranking attributes. Because data exploration is most meaningful for large datasets, we selected profiles with at least 1,000 matches for our evaluation. Our prototype operates on memory-resident data, and does all processing in memory. Due to a limitation in available RAM, we restrict our attention to users whose target profiles match up to 500,000 profiles. The chosen target profiles generated between 1,107 and 489,090 matches (median 102,492). Note that the size of the result set will often be reduced in practice by applying additional filtering criteria such as geographic distance between the seeker and the match, the freshness of the profile of the match, etc.

Parameters of the Algorithm. BARAC takes several parameters as input. The parameter N models the user's *attention span* – the number of items the user is likely to explore sequentially [20]. We used $N = 100$ for all experiments in this section. We modified the procedure **BuildGrid** (see Algorithm 2) by specifying an upper bound on the number of intervals per dimension, which we set to 5. This value is chosen according to *age*, the attribute with highest cardinality, and is set so that the values falling into a particular age interval are perceived as similar by a typical user. For most other dimensions, the domain cardinality is lower than 5, and so the upper bound is never reached, and the actual domain cardinality is used instead. We study the scalability of **BARAC** as a function of the dominance and quality thresholds θ_{dom} and θ_Q. There are no additional parameters in our formalism.

5.1 Scalability

In the first experiment, we ran **BARAC** for 100 users in the full space of 19 filtering attributes and 17 clustering attributes. Values of the dominance and quality thresholds were fixed at $\theta_{dom} = \theta_Q = 0.51$ for this experiment. Table 3 summarizes the median, average, minimum, and maximum run times of **BARAC**, with all times listed in milliseconds.

	Execution time(ms)			
	med	**avg**	**min**	**max**
BuildGrid	1756	2317	336	7814
Merge	13	23	6	119
Join	862	2912	258	37442
Total	3102	5499	600	40015

Table 3: Median, average, max and min processing times for $\mathcal{Q}_{topN}$ for 100 users, with $\theta_{dom} = \theta_Q = 0.51$.

According to Table 3, the median run time of **BARAC** is 3.1 sec, and the average run time is 5.5 sec. The run time of **BARAC** is dominated by **BuildGrid** and **Join**, as the execution time of **Merge** is negligible even in the worst case. While the maximum value for **Join** is quite high, motivating future performance optimizations, the run time reported in Table 3 may be unrealistically high. This is because the actual clustering dimensionality may be far lower than 17 for specific queries, and we further explore this in Section 5.3.

Figure 5(a) presents the run time of **BARAC** as the percentage of cases that completed under a certain time limit. **BARAC** completes in under 5 seconds in most cases, and takes longer than 10 seconds in only a handful of cases. In the remainder of this section we analyze the factors that contribute to the performance of **BuildGrid** and **Join**, and explore scalability as we vary the size of the dataset and the clustering dimensionality.

5.2 Varying Dataset Size

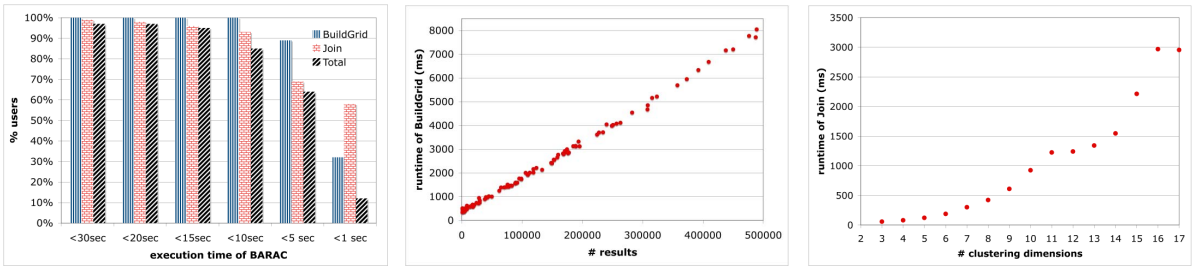
Figure 5(b) shows the performance of **BuildGrid** for 100 users as a function of dataset size – the number of items that pass the filtering conditions of the target profile. The data presented in this figure is the same as was used in Table 3 and Figure 5(a). Each data point corresponds to a particular target profile, and thus to a particular dataset size. All time measurements are in milliseconds. As before, the dominance and quality thresholds θ_{dom} and $\theta_{\mathcal{Q}}$ were both set to 0.51.

During **BuildGrid**, the seeker's filtering conditions are applied to memory-resident data in a single linear scan of the data, matches are identified, and a score is computed for every match. Items are then sorted on score in decreasing order. Finally, a data grid of matches is computed. We determined experimentally that score computation is the dominant factor in the execution time of **BuildGrid**. As Figure 5(b) demonstrates, the execution time of this stage increases linearly with the number of matches.

5.3 Varying Clustering Dimensionality

We now explore the impact of dimensionality on the performance of **Join**. For this experiment, we fix $\theta_{dom} = \theta_Q = 0.51$ and vary the dimensionality of the clustering space from 3 to 17. The first 3 clustering attributes are selected, and attributes are added one at a time in subsequent rounds. Attributes are added in the same order for all users, but this order was chosen randomly. We attempted several orders of adding attributes, and noticed no difference with respect to the performance trends of **Join**. We thus report our results with one particular order of adding attributes. Note that the filtering criteria and the scoring function are specified by the user's target profile, and are applied as before.

Figure 5(c) presents the execution time of **Join** as a function of



(a) **BARAC** as percentage of cases that completed under a certain time limit.

(b) **BuildGrid** as a function of dataset size.

(c) **Join** as a function of clustering dimensionality.

Figure 5: Run-time performance of BARAC.

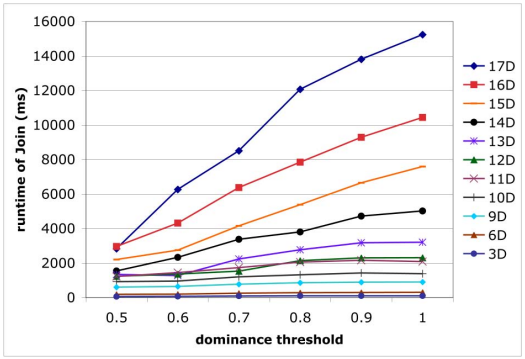


Figure 6: Join as a function of θ_{dom}.

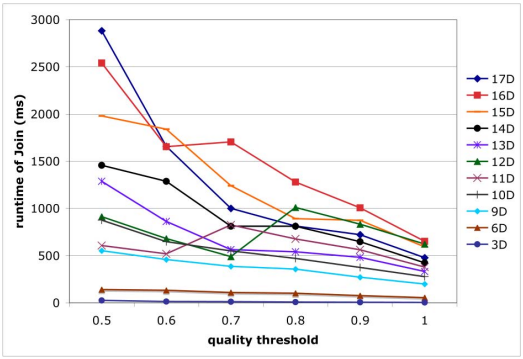


Figure 7: Join as a function of θ_Q.

clustering dimensionality. Each point is an average of execution times for the fixed dimensionality across all users. We observe that the execution time of **Join** increases with increasing dimensionality, but that it increases more significantly in some cases than in others. The general trend, with the exception of attributes 12, 13, and 17, discussed below, is that the execution time on **Join** increases approximately quadratically with increasing dimensionality.

Adding a clustering dimension to the set of dimensions *for a particular user* may not have an effect on the run time of the algorithm. For example, if we are adding the dimension *drinking*, but the user's filtering conditions are specifying a single value for this attribute, *drinking* = *no*, the attribute will not be added to the data grid, and so clustering will proceed as it did before the dimension was added. Attributes 12, 13 and 17 happen to be *marital status*, *wants more kids*, and *drinking* — low-cardinality attributes commonly restricted to a single value in the users' filtering conditions.

5.4 Varying Parameters of the Algorithm

Let us now see how the dominance and quality thresholds impact runtime performance. The dominance threshold θ_{dom} is used in **Merge**; the lower the threshold, the fewer intervals will **Merge** produce, and pass along to **Join**. Consequently, the run time of **Join** should increase as θ_{dom} increases. Figure 6 demonstrates that this trend holds for datasets of different dimensionality. Here, we fix $\theta_Q = 0.51$, and report the average run time of **Join** for each value of θ_{dom}, and for most dimensionality settings. A similar trend holds for the omitted dimensions, and we do not include them to make the chart more readable.

Varying the quality threshold θ_Q has the opposite effect on the run time of **Join**. This is because the higher the threshold, the fewer clusters are generated by **Join**, and the sooner it terminates. This intuition is supported by our experimental findings in Figure 7. Here, we set $\theta_{dom} = 0.51$ and present the average run time of **Join** for each value of θ_Q, and for most dimensionality settings. A similar trend holds for the omitted dimensions.

6. COMPARISON WITH CLIQUE

Having evaluated the effectiveness and efficiency of rank-aware clustering on Yahoo! Personals datasets, we now present the final set of experiments, in which we compare the clustering outcome achieved by BARAC with that of CLIQUE [2], a density-based subspace clustering algorithm that is rank-unaware. The goal is to demonstrate that treating the score, or rank, of each item as an additional attribute, and using it for clustering in a rank-unaware framework in insufficient. We used a dataset from Yahoo! Local that contains restaurant ratings by users with varying demographics. We decided to use this dataset to demonstrate wider applicability of our techniques.

6.1 Experimental Dataset

The Yahoo! Local dataset contains ratings of over 250,000 restaurants by Yahoo! users. The CLIQUE algorithm operates over ordered data and assumes that the domain of values of each attribute can be split into an equal number of equi-width histogram buckets. In order to make BARAC and CLIQUE comparable, we used 17 numerical attributes, each with five distinct values or value ranges. These attributes are described below.

A restaurant has an associated *price range*, on a scale of 1 (cheapest) through 5 (most expensive). Ratings are likewise assigned on a scale of 1 (lowest) through 5 (highest). A user who rates a restaurant has an associated age group and occupation. For each restaurant, we computed an average rating assigned to the restaurant by

CLIQUE	$\langle price \in [1,3]$ and $RTG_{26-30} = 1\rangle$ $\langle RTG_{18-25} \in [4,5]$ and $RTG_{61-100} \in [4,5]\rangle$ $\langle price \in [1,3]$ and $RTG_{18-25} \in [4,5]$ and $RTG_{26-30} \in [3,5]\rangle$
BARAC	$\langle price = 1$ and $RTG_{student} = 5$ and $RTG_{acad} = 5$ and $RTG_{18-25} = 5$ and $RTG_{26-30} = 5$ and $RTG_{41-45} = 5\rangle$ $\langle RTG_{31-35} = 5$ and $RTG_{51-60} = 5\rangle$ $\langle RTG_{31-35} = 5$ and $RTG_{41-45} = 5\rangle$

Table 4: Clusters generated by CLIQUE and BARAC.

users in each of the following eight age groups: $[18, 25]$, $[26, 30]$, $[31, 35]$, $[36, 40]$, $[41, 45]$, $[46, 50]$, $[51, 60]$, and $[61, 100]$. Similarly, we computed an average rating by users with each of the following eight occupations: *student*, *professional*, *academic*, *technology*, *sales*, *trade*, *unemployed*, and *retired*.

Our ranking function was *attribute-rank* with 5 score components, and included *price range*, from lower to higher, and ratings by users in the 18-25 and 26-30 age groups, and with *student* and *academic* occupation, from higher to lower. (See Section 4.1 for a detailed description of *attribute-rank*).

Both CLIQUE and BARAC operated over the full space of attributes, and over the entire dataset (that is, we did not specify any filtering conditions). BARAC was executed with the $\mathcal{Q}_{topN}$ clustering quality measure, $\theta_{dom} = 0.7$, $\theta_Q = 0.51$, and $N = 100$. These are the same settings as used in our user study in Section 4.2.

CLIQUE takes a density threshold θ_{dense} as input. We experimented with various values of θ_{dense}, and report results for $\theta_{dense} = 0.03$. This threshold was tuned manually to return a non-trivial number of interesting clusters and, in our judgment, corresponds to the best setting for our dataset.

6.2 Results

CLIQUE. Table 4 (line 1) lists a representative sub-set of clusters discovered by CLIQUE. The first cluster contains restaurants in the low to medium price-range that received the lowest possible rating from 26-30 year-olds. This cluster is not a good result for a user who is interested in restaurants with high ratings. The second cluster represents restaurants that were rated highly (with 4 or 5 stars) by users in the 18-25 and 61+ age groups, and represents a good clustering outcome, although it is not *tight* — the top-ranking restaurants in this cluster have a rating of 5 in both age groups. Similarly, the third cluster is not tight because the top-ranking restaurants in it have a rating of 5 in both age groups, and a price of either 1 or 2.

We also considered adding the rank of an item, computed based on item score, as an additional dimension for CLIQUE. However, this did not improve the clustering outcome for CLIQUE, due to the distribution of scores. Figure 8 presents the distribution of scores in our dataset, with the *attribute-rank* ranking function. Observe that the majority of items have low scores, and only 1% of the items have a score between 4 and 5. (Note that, while we report results with a single ranking function in this section, similar trends hold for *attribute-rank* with different ranking attributes, and with a different number of attributes.) Because few items have a high score, high-scoring items form a region of low density along the *score* attribute. Thus, if the density threshold is set too high, high-scoring items will not be part of the clustering result. An example of a cluster with low-scoring items is $\langle price \in [1,3]$ and $score \in [1,3]\rangle$.

Lowering the density threshold also does not improve clustering. This is because, as demonstrated in Figure 8, density increases with decreasing score. Therefore, if the density threshold is sufficiently low to include high-scoring items, it will also include low-scoring items, and so a density-based algorithm will merge together consecutive intervals along the score dimension. An example of a

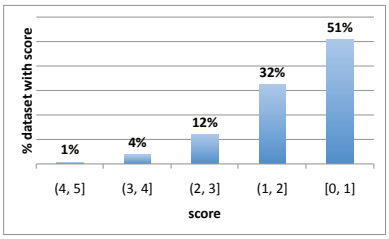


Figure 8: Score distribution with *attribute-rank*

cluster where this occurs is $\langle RTG_{tech} \in [1,5]$ and $score \in [1,4]\rangle$. This cluster was derived with $\theta_{dense} = 0.01$.

We observed that CLIQUE was very sensitive to the value of the density threshold θ_{dense}, with lower values of the threshold resulting in more clusters, but in clusters being less tight. Despite extensive manual turning of θ_{dense}, we were unable to find a threshold value that would produce tight clusters, or that would consistently place high-quality and low-quality items into separate clusters.

BARAC. Table 4 (line 2) lists a representative sub-set of clusters discovered by BARAC. All clusters are rank-aware and tight, addressing the problems observed for CLIQUE. We observe that there is some redundancy in terms of cluster descriptions, and items, in clusters two and three. Both clusters contain items that were rated highly by 31-35 year-olds. However, the two clusters are reported separately, because there does not appear to be sufficient agreement between the 41-45 and 51-60 age groups.

To conclude, we demonstrated that BARAC discovers rank-aware tight clusters, while CLIQUE does not discriminate between clusters of high-quality and low-quality items. Finally, note that, while the dataset used in this section contains only ordered attributes, BARAC also handles categorical attributes, as shown in Sections 4 and 5. On the other hand, CLIQUE only handles ordered attributes.

7. RELATED WORK

Subspace Clustering: Subspace clustering has been used extensively for data exploration in a variety of domains, see [15, 23] for reviews. CLIQUE [2] is the first bottom-up subspace clustering algorithm that relies on a global notion of *density* — the percentage of the overall dataset that falls within a particular subspace. ENCLUS [11] uses information entropy as the clustering objective, and shows a relationship between entropy and density, correlation, and coverage. Several extensions of the original algorithm were developed: MAFIA [22] creates an adaptive grid that takes into account the data distribution, CLTree [19] uses a decision-tree approach to identify high-density regions, while Cell-Based Clustering [8] improves scalability by partitioning the data so as to produce fewer clusters. We propose an algorithm that uses a rank-aware grid to determine locality, and relies on rank-aware clustering quality measures that are conceptually different from previous proposals.

Clustering Web Documents: The motivation for this work is similar to ours, namely, that grouping results and generating descriptions for these groups improves the user's ability to understand vast datasets. Clustering of text documents has been explored extensively in Information Retrieval (e.g., [6, 17]. Bonchi *et al.* [6] use search query logs to cluster results into coherent well-separated sets for presentation purposes. Leuski [17] experimentally demonstrates that presenting clusters of documents can be significantly more effective than presenting a ranked list. Our focus is on structured data, and on clustering in presence of ranking.

Clustering Relational Data: Li *et al.* [18] argue for native support of clustering and ranking operators in relational databases. In their framework, clustering and ordering attributes, and the number of clusters are specified by the user, and the focus is on efficiency. In addition to efficiency, we focus on automatically finding combinations of clustering attributes given a scoring function.

Ranking in Structured Datasets: Chaudhuri *et al.* [9] showed how correlations among attribute values can be used to automatically derive a ranking function. A related problem was also studied by Agrawal *et al.* [3]. Das *et al.* [12] considered rank-aware attribute selection, where the goal is to select a subset of ranking attributes that best reflect the ranking. In contrast, we assume that a ranking function, such as that discovered in [9], or a reduced ranking function such as that of [12], is given, and propose a *clustering* approach based on correlations between attributes and ranking.

Integrating Ranking with Clustering: A preliminary version of our work appeared in [24], and included motivation for rank-aware clustering and a qualitative analysis of clustering outcomes. Our current submission also includes a formalization of the problem, a description of a rank-aware clustering algorithm and an analysis of its complexity, and an extensive experimental evaluation. Sun *et al.* [25] presented RankClus, a framework that integrates ranking with clustering in a heterogeneous information network such as DBLP. RankClus is based on a mixture model that uses mutual reinforcement between clustering and ranking. Our high-level motivation is also to treat clustering and ranking as parts of a unified framework. However, our application domain — structured datasets with user-defined ranking functions, and technical approach are very different.

Diversification: Our work is related to result diversification in Web search [1, 4, 7, 16, 27], database queries [10, 26], and recommendations [5, 29]. Diversifying Web search results and recommendations aims to achieve a compromise between relevance and result heterogeneity, and the approaches do not rely on structured data. In [4], taxonomies are used to sample search results in order to reduce homogeneity. In [16], search results are clustered into groups of related topics that reflect different user interests. In recommendations [29, 14, 21], results are typically post-processed using pairwise item similarity in order to generate a list that achieves a balance between accuracy and diversity. While similarity functions can rely on structured data, only one list is returned, and it is not labeled. In the database context, Chen and Li [10] propose to post-process structured query results, organizing them in a decision tree for easier navigation. In [26], a hierarchical notion of diversity in databases is introduced, and efficient top-k processing algorithms are developed. While these approaches rely on structured data, they do not account for ranking.

8. CONCLUSION

In this paper we developed a clustering technique for the effective exploration of large structured datasets in the presence of ranking. Our approach is applicable to datasets with a large number of heterogeneous attributes. We formalized the *rank-aware interval-based clustering* problem and developed **BARAC**, a Bottom-up Algorithm for Rank-Aware Clustering. Our evaluation on datasets from a leading dating site showed that (i) users are more engaged with data exploration when presented with rank-aware clusters, and (ii) good run-time performance is achievable.

9. ACKNOWLEDGMENTS

We thank Duncan Watts and Jake Hoffman for valuable discussions during the initial stages of this project. We also thank Janet George, Tejaswi Kasturi, Tom Gulik, Prasenjit Sarkar, George Levchenko, Jacob Leatherman, and Tom Maher for their help with the implementation and release of the Yahoo! Find Love prototype. We thank Mor Naaman for his advise on user study design.

10. REFERENCES

[1] E. Agichtein, E. Brill, and S. T. Dumais. Improving web search ranking by incorporating user behavior information. In *SIGIR*, 2006.
[2] R. Agrawal et al. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD*, 1998.
[3] S. Agrawal et al. Automated ranking of database query results. In *CIDR*, 2003.
[4] A. Anagnostopoulos, A. Z. Broder, and D. Carmel. Sampling search-engine results. *WWW*, 2005.
[5] R. Boim, T. Milo, and S. Novgorodov. DiRec: Diversified recommendations for semantic-less collaborative filtering. In *ICDE*, 2011.
[6] F. Bonchi et al. Topical query decomposition. In *CIKM*, 2008.
[7] J. G. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*, 1998.
[8] J.-W. Chang and D.-S. Jin. A new cell-based clustering method for large, high-dimensional data in data mining applications. In *SAC*, 2002.
[9] S. Chaudhuri et al. Probabilistic ranking of database query resuls. In *VLDB*, 2004.
[10] Z. Chen and T. Li. Addressing diverse user preferences in sql-query-result navigation. In *SIGMOD*, 2007.
[11] C. H. Cheng et al. Entropy-based subspace clustering for mining numerical data. In *KDD*, 1999.
[12] G. Das et al. Ordering the attributes of query results. In *SIGMOD*, 2006.
[13] K. Järvelin and K. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM TOIS*, 20(4), 2002.
[14] J. A. Konstan. Introduction to recommender systems. In *SIGMOD*, 2008.
[15] H.-P. Kriegel et al. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *TKDD*, 3(1), 2009.
[16] K. Kummamuru et al. A hierarchical monothetic document clustering algorithm for summarization and browsing search results. In *WWW*, 2004.
[17] A. Leuski. Evaluating document clustering for interactive information retrieval. In *CIKM*, 2001.
[18] C. Li et al. Supporting ranking and clustering as generalized order-by and group-by. In *SIGMOD*, 2007.
[19] B. Liu et al. Clustering through decision tree construction. In *CIKM*, 2000.
[20] U. Manber et al. Experience with personalization of Yahoo! *Commun. ACM*, 43(8), 2000.
[21] S. M. McNee, J. Riedl, and J. A. Konstan. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI Extended Abstracts*, 2006.
[22] H. S. Nagesh. High performance subspace clustering for massive data sets. In *Master's thesis*. 1999.
[23] L. Parsons et al. Subspace clustering for high dimensional data: a review. *SIGKDD Explorations*, 6(1), 2004.
[24] J. Stoyanovich and S. Amer-Yahia. Rank-aware clustering for structured datasets. In *CIKM*, 2009.
[25] Y. Sun et al. RankClus: integrating clustering with ranking for heterogeneous information network analysis. In *EDBT*, 2009.
[26] E. Vee et al. Efficient computation of diverse query results. In *ICDE*, 2008.
[27] D. Xin et al. Extracting redundancy-aware top-k patterns. In *KDD*, 2006.
[28] G. Yang. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *KDD*, 2004.
[29] C. Yu, L. V. S. Lakshmanan, and S. Amer-Yahia. It takes variety to make a world: diversification in recommender systems. In *EDBT*, 2009.