# Rank-Aware Subspace Clustering for Structured Datasets

## Columbia University Computer Science Technical Report cucs-043-09

Julia Stoyanovich *
Columbia University
New York, NY, USA
jds1@cs.columbia.edu

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

## ABSTRACT

In online applications such as Match.com and Trulia.com users define structured profiles in order to find potentially interesting matches. Typically, profiles are evaluated against large datasets and produce thousands of matches. In addition to filtering, users also specify ranking in their profile, and matches are returned in the form of a ranked list. Top results in ranked lists are typically homogeneous, which hinders data exploration. For example, a user looking for 1- or 2-bedroom apartments sorted by price will see a large number of cheap 1-bedrooms in undesirable neighborhoods before seeing any apartment with different characteristics. An alternative to ranking is to group matches on common attribute values (e.g., cheap 1-bedrooms in good neighborhoods, 2-bedrooms with 2 baths). However, not all groups will be of interest to the user given the ranking criteria.

We argue here that neither single-list ranking nor attribute-based grouping is adequate for effective exploration of ranked datasets. We formalize rank-aware clustering and develop a novel rank-aware bottom-up subspace clustering algorithm. We evaluate the performance of our algorithm over large datasets from a leading online dating site.

## 1. INTRODUCTION

In online applications that involve large structured datasets, such as Yahoo! Personals and Trulia.com, there are often thousands of high-quality items, in this case, persons and apartments, that satisfy a user's information need. Users typically specify a structured target profile in the form of attribute-value pairs, and this profile is then used by the system to *filter* items. On dating sites, a target profile may specify the age, height, income, education, political affiliation, and religion of a potential match. In real estate applications, a profile describes a user's dream home by its location, size, and number of bedrooms. The number of matches to a specific profile is often very high, making *data exploration* an interesting challenge.

Typically users also specify ranking criteria which are used to *rank* matches. For example, in Yahoo! Personals, potential matches can be ranked by decreasing income or increasing age, while in Yahoo! Real Estate, available houses may be ranked by increasing price or decreasing size. Ranking helps users navigate the set of results by limiting the number of items that they see at any one time, and by making sure that the items users see first are of high quality (according to the ranking criteria). However, ranking also brings the disadvantage of *match homogeneity*: the user is often required to go through a large number of similar items before finding the next different item. This is illustrated in the following fictional example inspired by Yahoo! Personals.

EXAMPLE 1.1. *User* Mike *is looking for a date.* Mike *specifies that he is interested in women who are 20 to 30 years old and who have some college education, and requests that results be sorted on income in descending order. When inspecting the results,* Mike *notices that the top ranks are dominated by women in their late-twenties with a Master's degree. It takes* Mike *a while to scroll down to* the next set of matches *which are different from the top-ranking ones. In doing so, he skips over some unexpected cases such as younger women with higher education and income levels, or women with high income who did not graduate from college. After additional data exploration,* Mike *realizes that there is a* correlation *between age, education (filtering), and income (ranking). Such correlations would have been obvious if data were presented in* labeled groups *such as* `[20-24] year-olds with a bachelor's degree`, `[25-30] year-olds who make more than 75K`, *etc.*

A key point that arises from this example is that a user who is browsing a 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 understand general properties of a dataset, i.e., explore the data, particularly if the dataset is large.

The complexity of manual data exploration increases with more sophisticated ranking. For example, *Mike*'s profile could provide a custom scoring function that computes a score which is inversely proportional to the distance from his geographic location to the location of his match, and directly proportional to the match's income. Helping the users better understand the results, which enables easier navigation and profile refinement, is even more important in this case, given that correlations between item attributes and the ranking function are less obvious.

### 1.1 Motivating User Study

In order to better understand the challenges of ranked data exploration, we interviewed six potential users. All users were male, and they were all members of Yahoo! Re-

*Research supported in part by National Institute of Health grant 5 U54 CA121852-03

search. Users were asked to specify a realistic Yahoo! Personals profile, and to discuss their data exploration experience with us during the course of the interview. We conducted free-form interviews and did not use a questionnaire so as not to influence the users' statements by a limited menu of options. Users were allowed to select among a prespecified set of ranking attributes.

User profiles were evaluated by our prototype implementation, against the Yahoo! Personals dataset. Profile matches were presented in two ways: *ranked list* and *clustered items*. Three users were shown the *ranked list* first, followed by *clustered items*; the order was reversed for the other three users.

As the users were interacting with the *ranked list* interface, they noted that *results are homogeneous*: there was little variation in attribute values among top-ranked items. As a result, item attributes were ignored by most users, and the decision of whether to further explore a particular profile was based solely on the photo. All six users explicitly stated result homogeneity as a concern. The other key observation was that *ranking is opaque*. Indeed, item ordering was not considered helpful in data exploration and was not well-understood by some users. Even for single-attribute ranking, some users wanted to see items with a variety of values for the ranking attribute. Two users explicitly stated this limitation.

During their interaction with the *clustered items* interface users noted that *diverse results were more easily accessible*, and that *clustered items* helped them *refine their search preferences*. Three out of six users decided to refine their query moments after seeing clusters of results. Two of the three explicitly commented that the presentation enabled them to quickly understand the result set and to refine their query more effectively.

## 1.2 Limitations of Clustering Algorithms

Clustering is an effective data exploration method that is applicable to structured, semistructured, and unstructured data alike. Clustering algorithms assign $N$ items to $K \ll N$ groups, where $K$ is either known in advance or discovered by the algorithm. To be useful for data exploration, the algorithm must produce *meaningful descriptions* for the clusters. There are many families of clustering algorithms that can be used for this task. Some algorithms partition the dataset, while others assign each point to zero, one, or several clusters. Some algorithms operate over all item attributes, while others attempt *dimensionality reduction* techniques. In domains like Yahoo! Personals, where datasets are large and all items are described by a large number of attributes, it is intuitive to use *subspace clustering*.

Subspace clustering is an extension of traditional clustering that seeks to find clusters in different subspaces of a dataset [17]. Clusters of items are *high-quality* regions identified in multiple, possibly overlapping, subspaces. Many subspace clustering algorithms use the *density of a region* as a quality measure. In the simplest case, density is a percentage of data points that fall within a particular region, and the algorithm aims to find all regions that have density higher than a pre-defined threshold. We give an overview of one of the first subspace clustering algorithms, CLIQUE [1], in Section 3.1, and we illustrate it here with an example.

EXAMPLE 1.2. *Consider a fictional real estate example in Table 1: a database of 300 rental apartments, listing the number of bedrooms, number of bathrooms, size in $ft^2$, monthly rental price, and the number of such apartments currently on the market.* Mary *is looking for an apartment that is at least 600-$ft^2$ in size and has at least one bedroom, and she wants the matches sorted on price in increasing order. All apartments in Table 1 match* Mary*'s profile.*

| # beds | # baths | size ($ft^2$) | price ($) | # apts |
|---|---|---|---|---|
| 1 | 1 | 600 | 1800 | 5 |
| | 1.5 | 700 | 2100 | 55 |
| | 1 | 750 | 2900 | 25 |
| 2 | 1.5 | 700 | 2200 | 30 |
| | 1 | 800 | 2400 | 60 |
| | 2 | 800 | 2850 | 10 |
| | 2 | 950 | 3500 | 100 |
| 3 | 1.5 | 950 | 2900 | 5 |
| | 2 | 1000 | 3200 | 10 |

**Table 1: A fictional real estate database.**

Assume a density threshold $\theta = 0.1$. A typical density-based subspace clustering algorithm starts by dividing the range of values along each dimension (attribute) into cells, and by computing the density in each cell. For example, each distinct value of `#beds`, `size`, and `#baths` may correspond to a cell, and `price` may be broken down into intervals $(1500, 2000]$, $(2000, 2500]$, $(2500, 3000]$, *and* $(3000, 3500]$. Cells that do not pass the density threshold are pruned at this stage. The algorithm immediately prunes 600-$ft^2$ apartments ($\frac{5}{300} < \theta$), 750-$ft^2$ apartments ($\frac{25}{300} < \theta$), 1000-$ft^2$ apartments ($\frac{10}{300} < \theta$), and apartments in the $(1500, 2000]$ price range ($\frac{5}{300} < \theta$). Given *Mary*'s interest in cheaper apartments (price is her ranking condition), it is problematic that the cheapest apartments in the dataset, the 600-$ft^2$ apartments that cost \$1800, are pruned.

Next, the algorithm progressively explores clusters in higher dimensions by *joining* lower-dimensional ones. For example, the 1-dimensional cluster of 800-$ft^2$ apartments (70 items) can be joined with the 1-dimensional cluster of apartments in the $(2000, 2500]$ price range (145 items). The result of this join is a region with 60 800-$ft^2$ 2-bedrooms at \$2400 per month, which qualifies as a cluster since it passes the density threshold. However, the region that results from joining the 950-$ft^2$ apartments (105 items) with apartments in the $(2500, 3000]$ price range (40 items) does not qualify as a cluster (it contains only 5 items) and is pruned, losing the potentially interesting 3-bedrooms for a relatively low price (\$2900). Density decreases in higher dimensions and the algorithm stops when there are no more clusters to explore.

## 1.3 Challenges of Rank-Aware Clustering

A lower density threshold would evidently guarantee that some of the regions pruned using a higher threshold would be preserved. However, if the threshold is set too low, the algorithm would keep merging neighboring cells, ultimately identifying much larger clusters, and possibly one cluster containing the entire dataset. Moreover, not all regions that pass a typical clustering *quality metric*, e.g., density or entropy, are equally interesting to the user. Indeed, given a scoring function, some items, and hence some clusters, are more desirable than others (e.g., *Mary* has little interest in the 2-bedroom apartments that cost \$3500, but would like to

see the 1-bedrooms that cost $1800). Even when the density of a region is high, as is the case with 2-bedroom apartments for $3500, *Mary* would probably have less interest in them than in cheaper apartments. Therefore, we propose to explore *rank-aware clustering quality measures* which account for item scores and ranks in assessing cluster quality.

## 1.4 Paper Outline

In the remainder of this paper we present **BARAC**: a **B**ottom-up **A**lgorithm for **R**ank-**A**ware **C**lustering of structured datasets. We start by formalizing new clustering quality measures for rank-aware data exploration in Section 2, and develop an adaptation of a bottom-up APRIORI-style subspace clustering algorithm for this setting in Section 3. In Section 4 we present an extensive evaluation of the efficiency of **BARAC** on datasets from Yahoo! Personals, and show that our algorithm is efficient and scalable. We experimentally validate the effectiveness of our approach in Section 5, using both qualitative analysis, and results of a large-scale user study with a subset of Yahoo! Personals users.

# 2. FORMALISM

In this section we formalize rank-aware data exploration for structured datasets. We start by introducing the notion of a clustering quality measure, and then give the problem statement.

## 2.1 Regions and Clusters

We are given a dataset $\mathcal{D}$ where items are described by attribute-value pairs, including a special attribute *id* which uniquely identifies each item. Attributes belong to a set $\mathcal{A}$.

DEFINITION 2.1 (REGIONS). *A region $\mathcal{G}$ is a set of items labeled with a conjunction of predicates over attributes in $\mathcal{A}$, which, if evaluated on the dataset $\mathcal{D}$, results in all items in the region. The dimensionality of a region is simply the number of predicates that describe that region.*

A predicate specifies a value, or a range of values, for an attributes. Note that a region may be described by the predicate $P = \top$, in which case it evaluates to the entire dataset.

The predicates that describe a region are the *region description*. We will often use *region* and *region description* interchangeably. The following conjunction of predicates specifies a two-dimensional region:

$$\mathcal{G} : age \in [25, 30] \wedge education = Bachelor's$$

Any subset of predicates that define a region $\mathcal{G}$ is a *sub-region* of $\mathcal{G}$ ($\mathcal{G}$ is a *super-region* of its sub-regions.) A *cluster* is a region that satisfies a *clustering quality measure*.

DEFINITION 2.2 (CLUSTERING QUALITY MEASURE). *A clustering quality measure $\mathcal{Q}$ is a predicate over the distribution of items in a* region *$\mathcal{G}$ that makes $\mathcal{G}$ interesting to a user for the purpose of data exploration.*

In this paper we consider clustering quality measures that compare some statistic associated with the region to a threshold $\theta$. Let us now give some examples of measures that were developed in data mining. Given a region $\mathcal{G} = P_1 \wedge \ldots \wedge P_n$, we denote by $p(P_i)$ the proportion of items satisfying $P_i$ w.r.t. the entire dataset $\mathcal{D}$, i.e. $|P_i|/|\mathcal{D}|$.

A clustering quality measure may be stated with respect to the number of items in the region, as is the case with the *density measure* in CLIQUE [1], e.g., "a cluster is a region that contains at least $\theta$% of the total number of items":

$$\mathcal{Q}_{DENSE} : p(\mathcal{G}) \geq \theta \quad (1)$$

A clustering quality measure may encode *attribute correlation*,[1] i.e., higher-than-expected density of points, where the fraction of the observed number of items to the expected number is compared to a threshold $\theta$.

$$\mathcal{Q}_{CORR} : \frac{p(P_1 \wedge \ldots \wedge P_n)}{p(P_1) \times \ldots \times p(P_n)} \geq \theta \quad (2)$$

A clustering quality measure may specify *entropy*, with the intuition that regions with lower entropy have higher density and higher attribute correlation, as was shown in ENCLUS [7]:

$$\mathcal{Q}_{ENT} : H(\mathcal{G}) = H(P_1, \ldots P_n) \leq \theta \quad (3)$$

Given a dataset $\mathcal{D}$, a clustering algorithm returns a set of regions that satisfy a clustering quality measure.

## 2.2 Rank-Aware Clusters

In an on-line data exploration scenario, a user specifies a profile composed of filtering and ranking criteria. We assume that the user's filtering conditions result in a dataset $\mathcal{D}$ (and can thus take users out of the notation since we are interested in one user at a time). Ranking is expressed by a scoring function $\mathcal{S}$ which assigns a score *i.score* to each item $i \in \mathcal{D}$. We denote by $\mathcal{S}(\mathcal{D})$ the set of all items from $\mathcal{D}$ augmented with *i.score*. Typically, items are presented to the user as a single ranked list sorted by score.

We first argue that rank-unaware clustering measures (see Section 2.1) are inappropriate when users are interested in exploring ranked datasets.

EXAMPLE 2.1. *Consider user* Mary *from Example 1.2.* Mary *is interested in seeing apartments ranked by price in increasing order.* Ann*, another user who shares* Mary*'s filtering conditions, may be interested in seeing the same apartments sorted by size in decreasing order. Which clusters are best for which user depends on the user's ranking preferences. One reasonable option is to cluster apartments based on the scoring attribute. In particular,* Ann *may appreciate seeing the 950-$ft^2$ apartments which cost $2900 in the same cluster as the same-size apartments for $3500, while* Mary *may prefer to see 950-$ft^2$ apartments grouped together with the same-priced 750-$ft^2$ apartments. A subspace clustering measure that does not account for item scores would not distinguish between these two users, and would therefore be inappropriate for rank-aware data exploration.*

The score of each item can be treated as an additional attribute and can thus be used for clustering. Items can be clustered using a quality measure of the kind described in Section 2.1. However, as we argue in the following example, using scores as an additional clustering dimension still fails to effectively address data exploration for ranked datasets.

[1]Here and in the remainder of this paper we use "correlation" loosely, in the sense of "departure from independence" (*http://en.wikipedia.org/wiki/Correlation*); we do not account for the direction of the relationship between random variables.

EXAMPLE 2.2. *Consider again Example 1.2, where* Mary *wants to sort apartments by price. If item price is used as a clustering dimension, in the same way as other attributes, then* Mary *may see a high number of clusters, not all of which are of potential interest to her: e.g. a cluster of expensive 2-bedroom apartments may appear alongside a cluster of cheap 2-bedrooms. 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 intervals that are of high interest to* Mary *with those of low interest, resulting in a potentially large heterogeneous cluster with homogeneous results dominating the top ranks.*

Hence, we explore new clustering quality measures that use item scores and ranks to assess region quality.

DEFINITION 2.3 (RANK-AWARE CLUSTERING QUALITY). *A rank-aware clustering quality measure is a predicate over $\mathcal{S}(\mathcal{G})$ for a region $\mathcal{G}$ and a scoring function $\mathcal{S}$.*

Here, $\mathcal{S}(\mathcal{G})$ denotes the set of all items from $\mathcal{G}$ augmented with *i.score*. We explore different types of rank-aware quality measures, building on the assumption that users are more interested in clusters that contain items with high scores, and that they will only explore the best items in those clusters. We use $\mathcal{S}(\mathcal{G}, N)$ to denote $N$ highest scoring items in $\mathcal{S}(\mathcal{G})$. $N$ is a parameter in our formalism that models the user's *attention span* – the number of items the user is likely to explore sequentially [15]. This parameter can be customized per user, or it can be set to reflect the preferences of an average user.

The first measure, $\mathcal{Q}_{topN}$, states that a multi-dimensional region $\mathcal{G}$ is a cluster if it contains enough items that are in the top-$N$ of each of its one-dimensional sub-regions.

$$\mathcal{Q}_{topN} : \frac{|\mathcal{S}(\mathcal{G}, N) \cap \mathcal{S}(P_1, N) \cap \ldots \cap \mathcal{S}(P_m, N)|}{|N|} \geq \theta \quad (4)$$

$\mathcal{Q}_{topN}$ aims to discover attribute correlations among the high-scoring items in the dataset. We illustrate how this measure compares to density using Example 1.2. Recall that user *Mary* specified price as the ranking condition.

The join of the 700-$ft^2$ cluster with the $(2000, 2500]$ price range cluster preserves the lower-priced 1-bedrooms, since the top-$N$ items in the join correspond to the high-scoring items in the $(2000, 2500]$ cluster (one of the sub-regions) and to the high-scoring items in the 700-$ft^2$ cluster (its other sub-region). On the other hand, the join of 2-bathroom apartments with 950-$ft^2$ apartments would not contain any of the cheapest 2-bathroom apartments in its top-$N$ and would thus not qualify as a cluster.

$\mathcal{Q}_{topN}$ is a generalization of density from CLIQUE [1], where $N$ is substituted by $|\mathcal{D}|$, making the numerator equal to $|\mathcal{S}(\mathcal{G})|$, or simply $|\mathcal{G}|$.

The next measure, $\mathcal{Q}_{SCORE}$, states that a region is interesting if it contains high-scoring items in its top-$N$. $\mathcal{G}$ will have the highest-scoring items in its top-$N$ if the same high-scoring items are present in the top-$N$ lists of all of its one-dimensional sub-regions $P_1 \ldots P_k$. In the best case, the top-$N$ of the intersection of these regions will coincide with the top-$N$ of their union, which gives rise to the formula:

$$\mathcal{Q}_{SCORE} : \frac{\Sigma_{i \in \mathcal{S}(\mathcal{G}, N)} i.score}{\Sigma_{i \in \mathcal{S}(\cup_k P_k, N)} i.score} \geq \theta \quad (5)$$

$\mathcal{Q}_{SCORE}$ can be used to compare regions with a different number of items in their top-$N$ lists: a region with few high-scoring items in the top-$N$ may be of equal interest to the user as one with many lower-scoring items. Suppose that *Mary*'s scoring function is $\mathcal{S}_{Mary} : i.score = \frac{3500 - i.price}{3500 - 1800}$. Then, under $\mathcal{Q}_{SCORE}$, the region formed by the five 600-$ft^2$ 1-bedrooms has a quality score of five and is more interesting than the region formed by the ten 1000-$ft^2$ 3-bedrooms with a score of 1.76.

Finally, we present a measure that models the relationship between item scores and ranks. The intuition is that a region with exceptionally high-scoring items in high ranks may be just as interesting to the user as a region in which items have intermediate scores. We define this measure using NDCG (Normalized Discounted Cumulated Gain) [10], with $\mathcal{S}(\cup_k P_k, N)$ as the ideal vector.

$$\mathcal{Q}_{SCORE\&RANK} : AVG_{r \leq N} NDCG(\mathcal{S}(\mathcal{G}, N), \mathcal{S}(\cup_k P_k, N))[r] \quad (6)$$

Consider again Example 1.2 and *Mary*'s scoring function $\mathcal{S}_{Mary}$, and let us take $N = 5$. Let us compute the NDCG for the 1.5-bathroom apartments with the size of 950-$ft^2$. The ideal gain vector consists of scores of the 5 best items that either have 1.5 bathrooms or are 950-$ft^2$ in size, namely, the 1.5-bath 700-$ft^2$ apartments that cost \$2100 per month ($i.score = 0.82$). With $b = 2$ we derive: $DCG_{Ideal} = [0.82\ 1.64\ 1.55\ 1.64\ 1.77]$.

Let us now compute the NDCG for the 950-$ft^2$ 1.5-bath apartments. The top-5 list of this region consists of five 3-bedroom apartments at \$2900 ($i.score = 0.35$). We derive $DCG = [0.35\ 0.7\ 0.66\ 0.7\ 0.75]$. We now normalize each position in $DCG$ by the corresponding position in $DCG_{Ideal}$, average the values, and arrive at $NDCG = 0.43$.

## 2.3 Problem Statement

DEFINITION 2.4 (RANK-AWARE CLUSTERING). *Given a dataset $\mathcal{D}$, a scoring function $S$, a rank-aware clustering quality measure $\mathcal{Q}$ and an integer $N$,* find all clusters *in $\mathcal{D}$, i.e. regions of any dimensionality that satisfy $\mathcal{Q}$.*

# 3. RANK-AWARE SUBSPACE CLUSTERING

In this section, we give a brief overview of subspace clustering, formalize properties of our rank-aware subspace clustering algorithm, and finally present the algorithm.

## 3.1 Overview of Subspace Clusterings

We now give a general description of density-based bottom-up subspace clustering algorithms. The reader is referred to [17, 11] for comprehensive surveys.

Subspace clustering is a feature selection technique that aims to uncover structure in high-dimensional datasets [17]. Unlike Principal Component Analysis (PCA), where the goal is to identify the single best subset of features in which the dataset is then clustered, subspace clustering looks for multiple, possibly overlapping, subsets of features that are used to cluster different portions of the dataset.

Subspace clustering algorithms use several related quality measures, also referred to as clustering objectives, to guide the search. CLIQUE [1], one of the first algorithms in this family, relies on a global notion of *density*, which is simply the percentage of the overall dataset that falls within

a particular region. A later algorithm, ENCLUS [7], uses information entropy as the clustering objective.

CLIQUE operates in three steps to which we refer as **BuildGrid**, **Merge** and **Join**.

1. **BuildGrid** builds a histogram in each dimension, counting the number of points that fall within each bucket. For example, if the dimension is *age*, the outcome of this phase is a set of non-overlapping age intervals and the number of matches in each interval.

2. **Merge** neighboring histogram buckets (within the same dimension) that pass the density threshold; discard buckets that do not pass the threshold.

3. **Join** dimensions APRIORI-style. This step computes clusters in higher dimensions by joining lower-dimensional clusters (e.g., $age \in [25-30]$ with $income \in [50K-80K]$), and only keeping higher-dimensional clusters that pass the density threshold. This step relies on the *downward closure* property of the clustering quality metric to prune the search space.

Several extensions of the original algorithm were developed: MAFIA [16] creates an adaptive grid that takes into account the data distribution, CLTree [14] uses a decision-tree approach to identify high-density regions, while Cell-Based Clustering (CBF) [5] improves scalability by partitioning the data so as to produce fewer clusters.

## 3.2 Algorithm Properties

The **Merge** phase of our algorithm is different from the corresponding phase of density-based algorithms, and it relies on the notion of interval dominance with respect to a scoring function.

DEFINITION 3.1 (INTERVAL DOMINANCE). *Given a scoring function $\mathcal{S}$, an integer $N$, an attribute $a_i$, and any two consecutive value intervals $\mathcal{I}_1$ and $\mathcal{I}_2$ in the set of values from domain$(a_i)$, we say that $\mathcal{I}_1$* dominates *$\mathcal{I}_2$ w.r.t. $\mathcal{S}$ at top-$N$ iff $\mathcal{S}(\mathcal{I}_1, N) = \mathcal{S}(\mathcal{I}_1 + \mathcal{I}_2, N)$. We denote this as $\mathcal{I}_1 \prec_{S,N} \mathcal{I}_2$.*

Here, $+$ is simply the concatenation of two consecutive intervals. The intuition is that the top-$N$ items from the dominating interval are strictly better, w.r.t. the scoring function $\mathcal{S}$, than the items in the the top-$N$ of the dominated interval. For example, if $S$ ranks items in increasing order of age, then $\mathcal{I}_1 : age \in [25, 29]$ dominates $\mathcal{I}_2 : age \in [30, 34]$.

We refine this definition further. We say that $\mathcal{I}_1$ *dominates* $\mathcal{I}_2$ up-to a factor $\theta \in (0.5, 1]$, w.r.t. $\mathcal{S}$ at top-$N$ iff

$$\frac{|\mathcal{S}(\mathcal{I}_1, N) \cap \mathcal{S}(\mathcal{I}_1 + \mathcal{I}_2, N)|}{N} \geq \theta$$

We denote this by $\mathcal{I}_1 \prec_{S,N,\theta} \mathcal{I}_2$.

Consider again the intervals $\mathcal{I}_1 : age \in [25, 29]$ and $\mathcal{I}_2 : age \in [30, 34]$, and a scoring function that orders items on a combination of income and education: $\mathcal{S} = 0.25 * income + 0.75 * education$ (higher values of attributes *income* and *education* correspond to higher income and education levels, respectively). Because age positively correlates with income and with education, it is likely that $\mathcal{I}_1 \prec_{S,N=10,\theta=0.75} \mathcal{I}_2$.

We are interested in bottom-up clustering algorithms which build clusters in higher dimensions from lower-dimensional clusters. Such algorithms rely on the *downward closure* property, which allows for pruning of the search space, resulting in better runtime performance.

DEFINITION 3.2 (DOWNWARD CLOSURE). *We say that* downward closure *holds for a clustering quality metric $\mathcal{Q}$ iff, for any region $\mathcal{G}$, if $\mathcal{Q}$ holds over $\mathcal{G}$, then it also holds over every sub-region of $\mathcal{G}$.*

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 group $P_k$ with $N$ or more items, $\mathcal{Q}_{topN} = 1$. As dimensionality of the group increases, new sets are added to the intersection in the numerator of the expression. Thus the value of $\mathcal{Q}$ is strictly 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 region $\mathcal{S}(\mathcal{G}, N)$ consists of items that are either in the top-$N$ of all its one-dimensional sub-regions ($i \in \cap_k \mathcal{S}(P_k, N)$), or of items that have lower scores ($j \in \cap_k (\mathcal{S}(P_k) \backslash \mathcal{S}(P_k, N))$). The portion of $\cap_k \mathcal{S}(P_k, N)$ in $\mathcal{S}(\mathcal{G}, N)$ does not increase as more one-dimensional groups 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 non-increasing in dimensionality of the group. 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. Thus the values of $\mathcal{Q}_{SCORE}$ and $\mathcal{Q}_{SCORE\&RANK}$ are strictly non-increasing with increasing dimensionality.

## 3.3 Our Approach

Our proposed algorithm **BARAC**, **B**ottom-up **A**lgorithm for **R**ank-**A**ware **C**lustering, is an APRIORI-style algorithm with a flow that is similar to CLIQUE (Algorithm 1).

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

**Require:** dataset $\mathcal{D}$, scoring function $S$, $N$, $\theta_{\mathcal{Q}}$, $\theta_{dom}$, $maxBuckets$

1: $grid = \textbf{BuildGrid}(\mathcal{S}, \mathcal{D}, N, maxBuckets)$;
2: $mergedGrid = \textbf{Merge}(grid, N, \mathcal{S}, \theta_{dom})$;
3: $clusters = \textbf{Join}(mergedGrid, N, \mathcal{S}, \theta_{\mathcal{Q}})$;
4: **return** $clusters$

The procedure **BuildGrid** (Algorithm 2) starts by computing a score for each item $i \in \mathcal{D}$, 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. If $a_i$ is a categorical attribute with no natural ordering on its values (e.g. religion), a histogram bucket is created for each value $v_j \in domain(a_i)$. If $a_i$ is numerical (e.g., age) or ordinal categorical (e.g., body type), $domain(a_i)$ is broken down into at most $maxBuckets$ intervals of consecutive values. Bucket $j$ for attribute $i$ is denoted by $grid[i][j]$. Having established interval boundaries for attribute $a_i$ (lines 3-12), we assign to each interval the best $N$ items in $\mathcal{S}(\mathcal{D})$ from among those that fall within the range of the interval (lines 13-15).

**Merge** runs multiple passes of the procedure **OnePass-Merge** (Algorithm 3). **OnePassMerge** takes the $grid$ as input and expands the search space of the algorithm by

**Algorithm 2** Procedure **BuildGrid**

**Require:** $\mathcal{S}(\mathcal{D}), S, N, maxBuckets$
1: compute a list of items S(D), sorted 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: **if** $a_i$ is an unordered categorical attribute **then**
5: **for** $val_j \in domain(a_i)$ **do**
6: {allocate 1 column in $grid[i]$ per value $val_j$}
7: $grid[i][j].range = [val_j, val_j]$;
8: **end for**
9: **else**
10: divide $domain(a_i)$ into at most $maxBuckets$ consecutive intervals;
11: set $grid[i][j].range$ per interval;
12: **end if**
13: **for** each interval $j$ **do**
14: $grid[i][j].items = S(\sigma_{grid[i][j].range}(D), N)$;
15: **end for**
16: **end for**
17: **return** $grid$

considering, and possibly merging, runs of neighboring histogram buckets along the same dimension. Once the first run of **OnePassMerge** is done, it is invoked again on the output grid, and explores merging additional intervals.

The idea is that, for an attribute $a_i$, if neither of the two neighboring one-dimensional intervals $grid[i][j]$ and $grid[i][k]$ dominates the other (Definition 3.1), then it may be beneficial to consider their concatenation in the subsequent **Join** phase, in addition to considering both of them separately. This is because the top-$N$ items of the concatenated interval are sufficiently different from the top-$N$ items of the individual intervals, presenting additional clustering opportunities. If, however, one of the intervals dominates the other, then, by definition, the set of top-$N$ items of the concatenated interval is very similar (or exactly the same) as the top-$N$ items of the dominating interval, and so adding the concatenated interval to the search space is not helpful. We make two observations about **Merge**. First, the output grid is typically much larger than the original grid since all input intervals are also preserved in the result. Second, the lower the threshold $\theta_{dom}$, the fewer intervals are generated. We explore the impact of $\theta_{dom}$ on efficiency in Section 4.

**Algorithm 3** Procedure **OnePassMerge**

**Require:** $grid, N, \mathcal{S}, \theta_{dom}$
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** not $(grid[i][j] \prec_{S,N,\theta dom} grid[i][k]) \wedge$ not $(grid[i][k] \prec_{S,N,\theta dom} grid[i][j])$ **then**
7: {+ denotes interval concatenation.}
8: $grid[i][j+k].items = \mathcal{S}(grid[i][j].items \cup grid[i][k].items, N)$;
9: $addToGrid(mergedGrid[i], grid[i][j+k]$;
10: **end if**
11: **end for**
12: **end for**
13: **return** $mergedGrid$

The procedure **Merge** returns the $mergedGrid$, which contains all one-dimensional clusters. The procedure **Join**, which is invoked next, computes clusters in higher dimensions by progressively joining together lower-dimensional clusters. This procedure is the same as the corresponding procedure in CLIQUE [1], and we describe it here for completeness using our terminology.

**Algorithm 4** Procedure **doJoin**

**Require:** $Clusters_{K-1}$, $\theta_{\mathcal{Q}}$
1: $Regions_K = \emptyset$;
2: **for** $\mathcal{C}_1 \in Clusters_{K-1}$ **do**
3: **for** $\mathcal{C}_2 \in Clusters_{K-1}$ **do**
4: **if** $compatible(C_1, C_2)$ **then**
5: $append(Regions_K, joinClusters(C_1, C_2))$;
6: **end if**
7: **end for**
8: **end for**
9: $Clusters_K = prune(Regions_K, \theta_{\mathcal{Q}})$;
10: **return** $Clusters_K$;

**Join** repeatedly invokes the sub-routine **doJoin** and terminates when no more clusters are identified. Procedure **doJoin**, presented in Algorithm 4, takes $(k-1)$-dimensional clusters and a quality threshold as input, and returns a set of $k$-dimensional clusters. This is done by first identifying a candidate set of $k$-dimensional regions (lines 2-8), and then pruning the set by removing all regions that do not pass the quality threshold $\theta_{\mathcal{Q}}$ (line 9). We omit pseudo-code for some of the subroutines, but describe them verbally below.

Assume that the relation $<$ represents a lexicographic ordering on attribute names. Assume also that a cluster $\mathcal{C}$ is represented by a set of *intervals*, with the number of intervals in the set corresponding to the dimensionality of the cluster. Each interval records the attribute name (e.g., *age* or *income*), and the low and high values that specify the range. So, an interval $age \in [25, 29]$ has $attribute = age$, $low = 25$, and $high = 29$. Two intervals are considered equal if they reference the same attribute name and the same range of values.

Two $(k-1)$-dimensional clusters $\mathcal{C}_1$ and $\mathcal{C}_2$ are said to be *compatible* if they contain $k-2$ equal intervals, and if the $(k-1)^{st}$ interval of $\mathcal{C}_1$ is lexicographically lower than the $(k-1)^{st}$ interval of $\mathcal{C}_2$. The result of **joinClusters**$(\mathcal{C}_1, \mathcal{C}_2)$ is a $k$-dimensional region described by the union of the intervals of $\mathcal{C}_1$ and $\mathcal{C}_2$.

The quality measure $\mathcal{Q}$ can be any one of the measures defined in Definition 2.3. During the **Join** step, all measures are applied to $\mathcal{S}(\mathcal{G}, N)$, the top-$N$ items of each region $\mathcal{G}$. We compute the top-$N$ lists for each grid interval in line 14 of Algorithm 2. As intervals are merged, and as clusters are joined to produce higher-dimensional clusters, top-$N$ lists are re-computed (line 9 of Algorithm 3).

In the worst case **Join** will explore all combinations of dimensions. However, this worst case is very coarse. Actual run-time performance is highly data-dependent, as we show in the next section. **Join** terminates when there are no more pairs of compatible clusters which satisfy the quality threshold. This is guaranteed by the downward closure property (see Definition 3.2). The lower the value of $\theta_{\mathcal{Q}}$, the higher the number of clusters generated by our algorithm. We explore the impact of different threshold values on the run time performance in the next section.

| | |
|---|---|
| **# users** | 100 |
| **# filtering attrs** | 3-15, median 5 |
| **# ranking attrs** | 1-6, median 3 |
| **dataset size** | 1,107 - 489,090, median 102,492 |

**Table 2: Characteristics of target profiles.**

# 4. EVALUATION OF PERFORMANCE

We implemented **BARAC** with our three clustering quality measures (Section 2.2.) Since $\mathcal{Q}_{SCORE}$ behaved similarly to $\mathcal{Q}_{topN}$, we only report results with the latter and $\mathcal{Q}_{SCORE\&RANK}$. Our prototype is implemented in Java and operates on memory-resident data. All experiments were executed on a 64-bit machine with two Intel Xeon 2.13GHz processors and 4GB of RAM, running RedHat EL AS 4.

## 4.1 The Yahoo! Personals Dataset

**Dataset.** We evaluated the performance of **BARAC** on a dataset from 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, e.g., age, height, occupation, education, income, etc. 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.

For the purpose of our experiments we focus on computing matches for male users, as there are at least one order of magnitude more males searching for females. We store a snapshot of *target profiles* of male users whom we call *seekers*, and of *personal profiles* of female users. The snapshot is as of a recent month in 2008, and contains all profiles that were registered with the dating service up to and including that month. 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.

**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. Table 2 summarizes the characteristics of the selected target profiles. 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. 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.

**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.

| | |
|---|---|
| **filtering** | *numerical*: age, height.<br>*ordinal categorical*: body type, education level, income, religious services (attendance frequency).<br>*categorical*: gender, sexual orientation, ethnicity, eye color, hair color, smoking, drinking, marital status, have kids, want more kids, employment status, profession, personality type, religion, political views. |
| **ranking** | age, height, body type, education level, income, religious services. |
| **ignored** | location, living situation, social personality, TV watching habits, languages, sense of humor, interests, love style, astrological sign. |

**Table 3: Structured attributes in the dating dataset.**

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.

$$geo\_rank = \frac{attribute\_rank}{1 + (geo\_distance/100)} \tag{7}$$

We will discuss in detail in Section 5.3 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 Scalability

In the first part of our experiments, we study the behavior of **BARAC** with the $\mathcal{Q}_{topN}$ quality measure, and the

| | 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 4: Median, average, max and min processing times for $\mathcal{Q}_{topN}$ for 100 users, with $\theta_{dom} = \theta_Q = 0.5$.**

*attribute-rank* scoring function. We analyze performance in terms of three distinct stages: **BuildGrid**, **Merge**, and **Join**. See Section 3.3 for a description of these stages.

**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 [15]. We used $N = 100$ for all experiments in this section. $maxBuckets$, used by the procedure **BuildGrid** (see Algorithm 2), specifies an upper bound on the number of intervals per dimension. We set it 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 $\theta_{dom}$ and $\theta_Q$. There are no additional parameters in our formalism.

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 $\theta_{dom} = \theta_Q = 0.5$ for this experiment. Table 4 summarizes the median, average, minimum, and maximum run times of **BARAC**, with all times listed in milliseconds.

According to Table 4, the median run time of **BARAC** is 3.1 seconds, and the average run time is 5.5 seconds. The run time of **BARAC** is dominated by **BuildGrid** and **Join**, while 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 4 may be unrealistically high. This is because, as discussed in Section 4.1, the actual clustering dimensionality is far lower than 17 for specific queries.

Figure 1 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.

### 4.2.1 Varying Dataset Size

Figure 2 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 4 and Figure 1. 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 $\theta_{dom}$ and $\theta_Q$ were both set to 0.5.

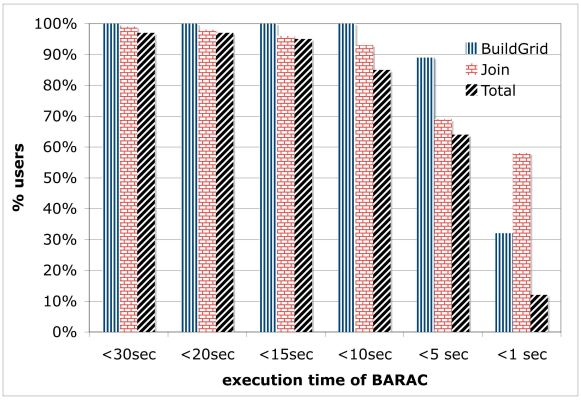


**Figure 1: Performance of BARAC as percentage of cases that completed under a certain time limit.**

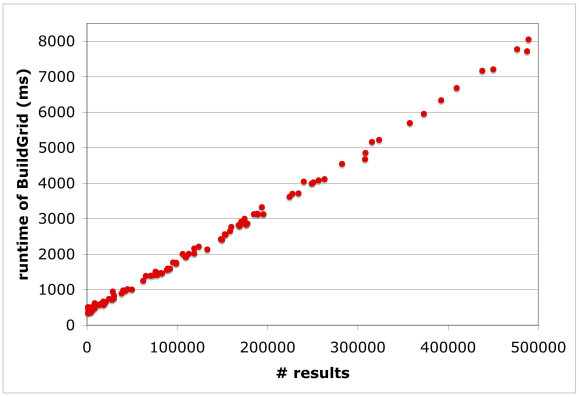


**Figure 2: Runtime performance of the *BuildGrid* stage as a function of dataset size.**

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 2 demonstrates, the execution time of this stage increases linearly with the number of matches.

### 4.2.2 Varying Clustering Dimensionality

We now explore the impact of dimensionality on the performance of **Join**. For this experiment, we fix $\theta_{dom} = \theta_Q = 0.5$ 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 have 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.

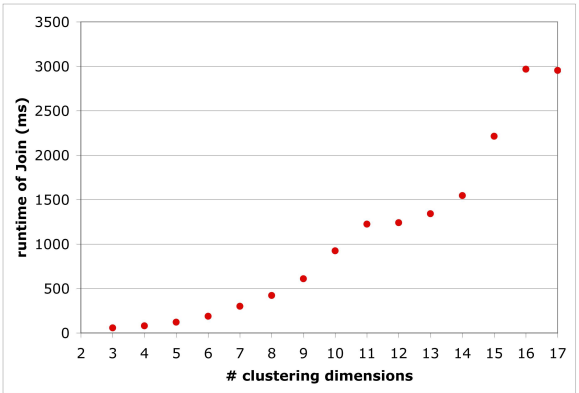


**Figure 3: Execution time of Join as a function of clustering dimensionality.**

Figure 3 presents the execution time of **Join** as a function of clustering dimensionality. Each point is an average of execution times for the fixed dimensionality across all users. We observe that the average execution time of **Join** increases as the dimensionality increases, but that it increases more significantly in some cases than in others. The general trend, with the exception of attributes 12, 13, and 17, which we discuss below, seems to be 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 or 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*. These are all low-cardinality attributes, which users commonly restrict to a single value in their filtering conditions.

### 4.2.3 Varying Parameters of the Algorithm

Let us now see how the dominance and quality thresholds impact runtime performance. The dominance threshold $\theta_{dom}$ 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 $\theta_{dom}$ increases. Figure 4 demonstrates that this trend holds for datasets of different dimensionality. Here, we fix $\theta_Q = 0.5$, and report the average run time of **Join** for each value of $\theta_{dom}$, and for most dimensionality settings.

Varying the quality threshold $\theta_Q$ 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 5. Here, we set $\theta_{dom} = 0.5$ and present the average run time of **Join** for each value of $\theta_Q$, and for most dimensionality settings.

In fact, for higher values of the quality thresholds it is often the case that no clusters at all exist of sufficiently high quality. When this happens, **Join** terminates after its initial round of processing, in which it attempts to join 1D intervals into 2D clusters. In Figure 6 we plot the percentage of users for whom clusters were identified, as a function of the quality threshold $\theta_Q$. We plot the same data here as in Figure 5, with the same dominance threshold setting, $\theta_{dom} = 0.5$.

# 5. EVALUATION OF EFFECTIVENESS

## 5.1 Qualitative Evaluation

We now give a qualitative intuition of the kinds of clusters that are discovered by **BARAC**. We use the $Q_{topN}$ quality metric, and the *attribute-rank* ranking function for the purpose of this evaluation. We focus our attention on the following set of filtering conditions that are in-line with our motivating example in Section 1: *age* $\in$ *[25,35], height* $\in$ *[160cm, 175cm], education* $\geq$ *Bachelor's, ethnicity = Caucasian, body type* $\in$ {*slim, slender, average, athletic, fit*}.

This set of filtering conditions returns over 30,000 matches. We rank the matches on a combination of *income* and *education*, both from higher to lower. There are about 100 top-matches: women with post-graduate education who make more than \$150K. About two thirds of the top matches are over 29 years old, and so younger matches would not be easily accessible if results were returned as a ranked list.

Let us now cluster the results of **BARAC**, using the $Q_{topN}$ quality metric. The following are some of the clusters that are returned and that deal directly with the correlation between income and age, and income and education: $age \in [25, 27] \wedge income \in [\$35K, \$75K]$, $age \in [28, 33] \wedge income \in [\$75K, \$150K]$, $age \in [28, 33] \wedge income \geq \$150K$, $age \in [31, 35] \wedge income \geq \$75K$, $age \in [28, 33] \wedge education =$ *post graduate*, $age \in [31, 35] \wedge education =$ *post graduate*.

Note that two clusters are returned that contain different sets of matches with age between 28 and 33. Note also that the younger matches, aged between 25 and 27, are returned as a cluster with relatively lower income. These matches would not have been easily accessible in a single ranked list. **BARAC** also returned several clusters that are not directly related to the ranking conditions, but for which a correlation was detected among attributes at top ranks. So, there was a cluster of matches who are politically *very conservative* or *conservative* and who attend religious services *more than once a week* or *weekly*. Another cluster consisted of matches who are politically *middle of the road* or *liberal* and who attend religious services no more often than *monthly*.

## 5.2 User Study

We evaluated the effectiveness of **BARAC** on the Yahoo! Personals dataset, by inviting a subset of registered users to participate in the experiment. Our experiment was implemented as part of Yahoo! Research Sandbox, and is available to registered Yahoo! Personals users at `findlove.sandbox.yahoo.com`. Like Yahoo! Personals, we use location-based filtering (e.g., *find matches who live within 250 miles of Austin, TX*), and provide 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. Section 4.1 describes the dataset. In our study we implemented the *attribute-rank* ranking function.

Our users are accustomed to ranked lists, which differ from BARAC groups in two ways. First, the set of results may be different. Secondly, results are presented in labeled groups. Our user study was designed so as to isolate the

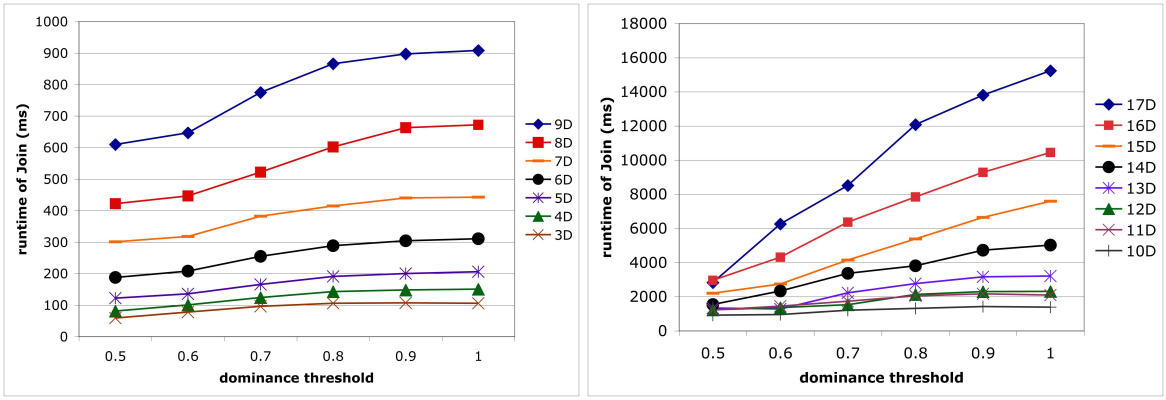


**Figure 4: Execution time of Join as a function of $\theta_{dom}$.**

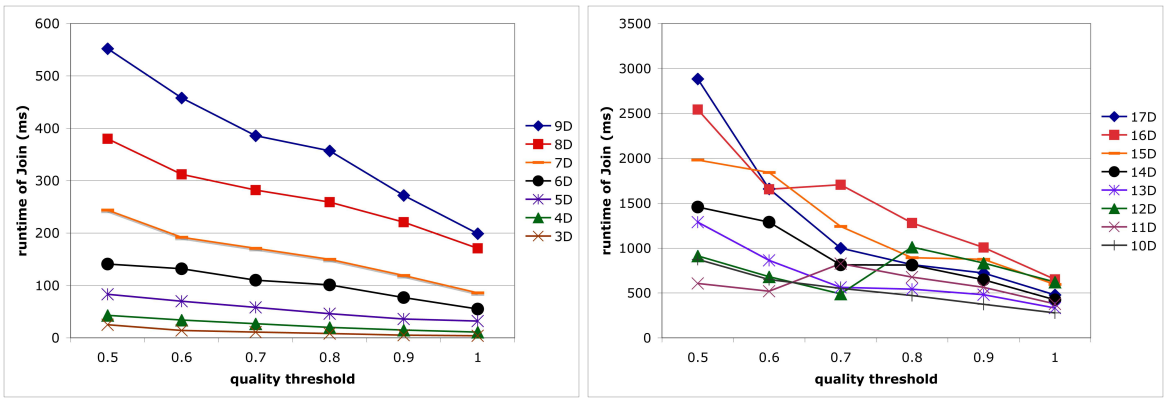


**Figure 5: Average execution time of Join as a function of $\theta_Q$.**

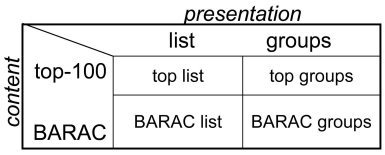


**Figure 7: User study design matrix.**

effects of these two aspects. Figure 7 summarizes the design of our study, in which we compare four treatments.

The *top list* treatment shows the top-100 matches in a traditional ranked list. Figure 8 presents a screenshot of the *top list* interface.

The same interface was used for the *BARAC list* treatment. Figure 9 shows the *BARAC groups* interface. The interface for the *top groups* treatments is similar, but with group labels of the kind *Top 1-10*, *Top 11-20*, etc.

*BARAC groups* is our rank-aware clustering, see Figure 9 for a screenshot. Matches are clustered and 10 groups are chosen and presented. The user may expand a group to see its content.

BARAC will often generate more than 10 groups, which requires group selection. We considered several group selection heuristics, and decided to choose groups so as to *maximize diversity of group descriptions*. Other heuristic, such as maximizing total items score, or minimizing item overlap between groups, are also possible, and we leave an in-depth

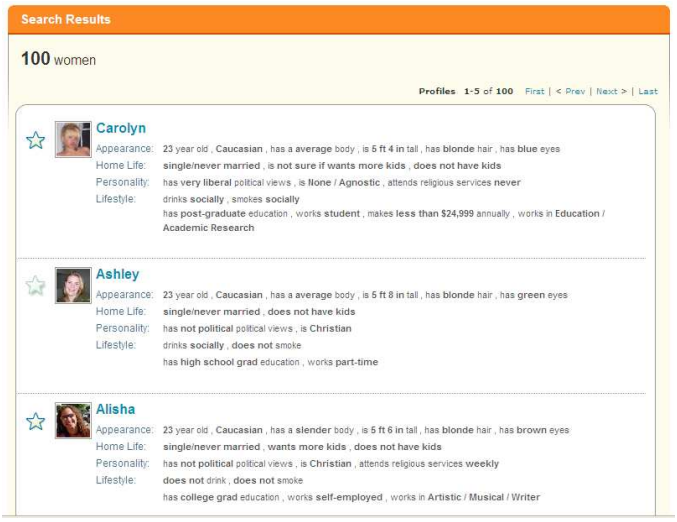


**Figure 8: The *top list* and *BARAC list* interface.**

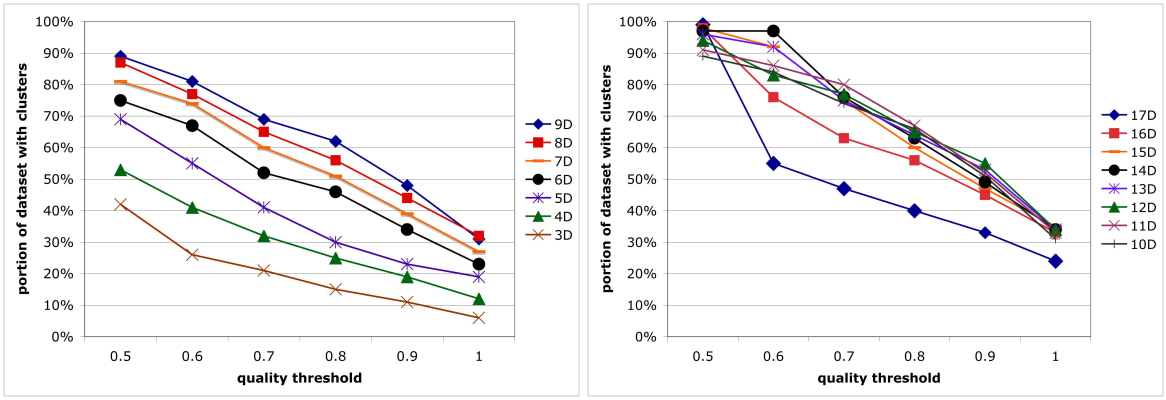


**Figure 6: Percentage of users for whom BARAC identified clusters, as a function of $\theta_Q$.**

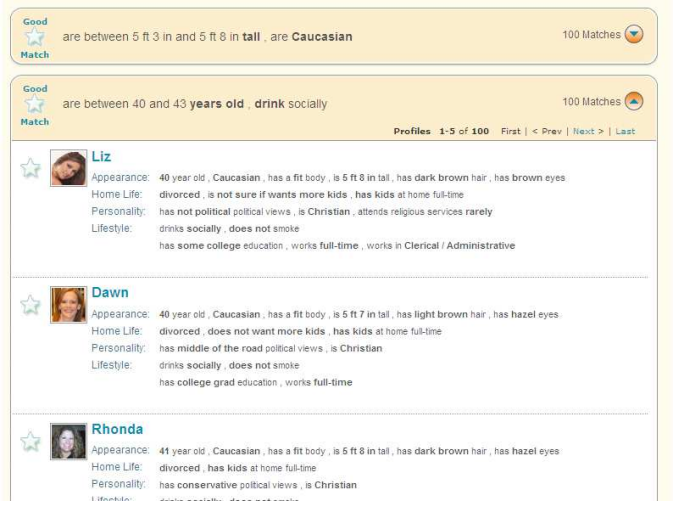


**Figure 9: The *BARAC groups* interface.**

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

**Table 5: 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 6: Number of faves per search.**

study of group selection to future work. Our group selection algorithm uses K-means to identify 10 clusters of groups with similar descriptions. Similarity between groups accounts for both attribute names and attribute value ranges. Having identified 10 clusters, we select a representative from each cluster that has the highest total score up to top-10. The chosen representatives are presented to the user.

*top groups* isolates the effect of grouping, with data coming from the ranked list. The user is presented with 10 groups of 10 results each. The first group correspond to the first 10 matches, followed by a group containing the next 10, etc. The interface is similar to that for *BARAC groups* (see Figure 9), with the exception that group labels are *Top 1-10*, *Top 11-20*, etc.

*BARAC list* presents a ranked list of matches produced by *BARAC groups*. 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*, see Figure 8.

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 too-few results or scores which are 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 (see Figures 8 and 9).

**Results.** The first metric we use to compare our 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 5 presents 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 group descriptions 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*, and

the difference in the number of searches with faved groups is statistically significant (Pearson's $p < 0.05$).

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 6 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.

## 5.3 Choosing a Clustering Quality Metric

We now demonstrate the qualitative difference between two proposed quality measures, $\mathcal{Q}_{topN}$ and $\mathcal{Q}_{SCORE\&RANK}$. We use the profile of one particular user, whom we call $user_1$, as an example in this section. $user_1$ is a representative user with about 60,000 profile matches.

As we discussed in Section 2, $\mathcal{Q}_{topN}$ favors regions that contain many items that are in the top-$N$ lists of their sub-regions, irrespective of the scores and ranks of those items in top-$N$ lists of the sub-regions. Conversely, $\mathcal{Q}_{SCORE\&RANK}$ assigns a higher quality score to a region in which top-$N$ lists of the sub-regions 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 scoring function. $\mathcal{Q}_{topN}$ treats all items with $N$ highest scores equally, and is thus appropriate for scoring 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 scoring function *attribute-rank* is one such function.

Conversely, $\mathcal{Q}_{SCORE\&RANK}$ is 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 scoring function *geo-rank* is one such function. We plot the distribution of the top-100 scores of $user_1$'s items for the two ranking function in Figure 10.

We now demonstrate that $\mathcal{Q}_{SCORE\&RANK}$ is more appropriate to use in conjunction with the *geo-rank* scoring function for $user_1$. We fix $\theta_{dom} = 0.5$ and $\theta_Q = 0.7$, and compare the sets of clusters that were identified by $\mathcal{Q}_{topN}$ and $\mathcal{Q}_{SCORE\&RANK}$. $\mathcal{Q}_{topN}$ identified 11 clusters, while $\mathcal{Q}_{SCORE\&RANK}$ identified 33 clusters. One of the clusters returned by $\mathcal{Q}_{topN}$ was not returned by $\mathcal{Q}_{SCORE\&RANK}$, we call it $\mathcal{G}_{topN}$. $\mathcal{Q}_{SCORE\&RANK}$ found 23 clusters that were not found by $\mathcal{Q}_{topN}$. We refer to the highest- and lowest-quality clusters from this set as $\mathcal{G}^+_{SCORE\&RANK}$ and $\mathcal{G}^-_{SCORE\&RANK}$. We summarize some properties of $\mathcal{G}_{topN}$, $\mathcal{G}^+_{SCORE\&RANK}$ and $\mathcal{G}^-_{SCORE\&RANK}$ in Table 7, where we compare the top-$N$ list of each cluster to the ideal top-$N$ list in terms of ranks and scores. The value in the column "score loss at 10" contains the difference between the total scores of the top-10 items from the ideal (the union of top-$N$ items of the individual predicates forming each cluster), and the top-10 items for the cluster.

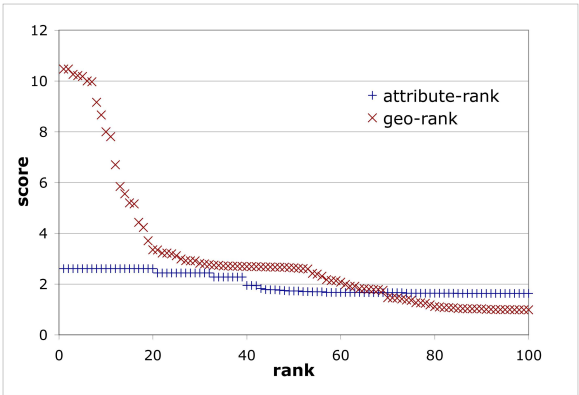


**Figure 10: Top-100 scores for *attribute-rank* and *geo-rank* for $user_1$.**

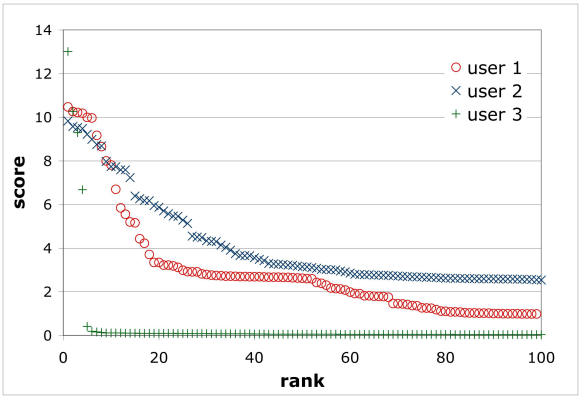


**Figure 11: Top-100 scores for *geo-rank* for 3 users.**

$\mathcal{G}_{topN}$ is a two-dimensional cluster on attributes *height* and *income*. The top-$N$ list of $\mathcal{G}_{topN}$ contains items that were in ranks 1 through 47 in the top-$N$ list on *height* (median rank 27), but in ranks 8 through 100 on *income* (median rank 70). So, while the intersection happens at the top of the top-$N$ on *height*, it is closer to the bottom of the top-$N$ on income. This cluster seems less valuable than the other two clusters in the table, based on both the lower score loss, and the lower (worse) median rank.

$\mathcal{Q}_{SCORE\&RANK}$ is sensitive to the distribution of scores. The same scoring 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 11. 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

| | best rank | worst rank | median rank | score loss at 10 |
|---|---|---|---|---|
| $\mathcal{G}_{topN}$ | 8 | 100 | 70 | 1.77 |
| $\mathcal{G}^{+}_{SCORE\&RANK}$ | 1 | 98 | 40 | 5.61 |
| $\mathcal{G}^{-}_{SCORE\&RANK}$ | 1 | 99 | 40 | 5.42 |

**Table 7: Characteristics of some groups that were identified using $\mathcal{Q}_{topN}$ and $\mathcal{Q}_{SCORE\&RANK}$.**

for $\mathcal{Q}_{SCORE\&RANK}$, and a score-insensitive quality measure like $\mathcal{Q}_{topN}$ should be used instead.

# 6. RELATED WORK

**Clustering Web Documents**: The motivation for this work is similar to ours, namely, that grouping results and generating descriptions for these groups greatly improves the user's ability to understand vast datasets. Clustering of text documents has been explored extensively in Information Retrieval [12, 8, 4]. Leuski [12] experimentally demonstrates that presenting clusters of documents can be significantly more effective than presenting a ranked list. He also shows that clustering can be as effective as the interactive relevance feedback approach based on query expansion. In [8] the authors combine an offline (query-independent) document clustering method and an online (query-dependent) method to generate multi-document summaries of clustered news articles. Bonchi et al [4] use search query logs to cluster search results into coherent well-separated sets for presentation purposes. In contrast, our work focuses on the interaction between structure and ranking in clustering.

**Clustering Relational Data**: Li et al [13] argue for native support of clustering and ranking operators in relational databases. The authors demonstrate how clustering can be implemented by means of a bitmap index over a summary grid. In their framework, the grouping (clustering) and ordering attributes, as well as the number of clusters, are specified by the user, and the focus of the work is on efficiency. The focus of our work is, in addition to efficiency, on automatically determining which clustering attributes are meaningful given a scoring function.

**Faceted Navigation:** This methods facilitates information discovery over large datasets. Faceted classification defines items using mutually exclusive, collectively exhaustive properties [21], and has been used in faceted navigation, where items are classified simultaneously along multiple facets, and item counts are presented per facet. A strong limitation is that a facet is a single attribute, and no attribute correlations are captured. Several extensions of the faceted data model were proposed. Ross and Janevski [18] developed *entity algebra*, a faceted query language that supports operators such as selection and semi-join. Ben-Yitzhak et al [3] extended faceted navigation to include quantitative summary information other than item counts, e.g. average price and rating, and developed methods for efficient computation of such statistics over correlated facets.

Roy et al [19] proposed techniques which dynamically suggest facets for database exploration, with the goal of minimizing navigation time. At every step, the system asks the user a set of questions about his information need, and dynamically fetches the next most promising set of facets. In [9], the authors extend Solr, a popular search engine, with dynamic faceted search for exploring data with both textual content and structured attributes. Given a keyword query, the system selects a small set of interesting attributes, where interestingness measures the difference between expected and actual attribute values.

Our work is complementary to faceted navigation. While we focus on attribute correlations, our analysis can be used to propose to the user which, among many attributes, may be more interesting to explore in a particular dataset.

**The Many-Answers Problem:** In [6] the authors state the *Many-Answers Problem* and show how correlations among attribute values in a structured datasets can be used to automatically derive a suitable ranking function. To this end, the authors develop a comprehensive probabilistic information retrieval ranking model and present efficient processing techniques. A related *Empty-Answers Problem* is discussed in [2]. Here, the authors present an adaptation of *inverse document frequency* (IDF) that is used for automatic ranking of results. The authors also propose to incorporate workload information into the ranking.

**Integrating Ranking with Clustering:** Sun et al [20] recently 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.

# 7. CONCLUSION

In this paper we introduced rank-aware clustering, a novel result presentation method for large structured datasets. We developed rank-aware clustering quality measures, and proposed BARAC: a Bottom-up Algorithm for Rank-Aware Clustering, an Apriori-style algorithm geared specifically at such quality measures. We presented an extensive experimental evaluation of the scalability of our approach, and demonstrated its effectiveness with a large-scale user study.

# 8. ACKNOWLEDGMENTS

We would like to thank Duncan Watts and Jake Hoffman for valuable discussions during various 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. Finally, we thank Mor Naaman for his advise on user study design.

# 9. REFERENCES

[1] R. Agrawal et al. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD*, 1998.

[2] S. Agrawal et al. Automated ranking of database query results. In *CIDR*, 2003.

[3] O. Ben-Yitzhak et al. Beyond basic faceted search. In *WSDM*, 2008.

[4] F. Bonchi et al. Topical query decomposition. In *CIKM*, 2008.

[5] 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.

[6] S. Chaudhuri et al. Probabilistic ranking of database query resuls. In *VLDB*, 2004.

[7] C. H. Cheng et al. Entropy-based subspace clustering for mining numerical data. In *KDD*, 1999.

[8] W. Dakka and L. Gravano. Efficient summarization-aware search for online news articles. In *JCDL*, 2007.

[9] D. Debabrata et al. Dynamic faceted search for discovery-driven analysis. In *CIKM*, 2008.

[10] K. Järvelin and K. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM TOIS*, 20(4), 2002.

[11] H.-P. Kriegel et al. Detecting clusters in moderate-to-high dimensional data: Subspace clustering, pattern-based clustering, and correlation clustering. In *VLDB*, 2008.

[12] A. Leuski. Evaluating document clustering for interactive information retrieval. In *CIKM*, 2001.

[13] C. Li et al. Supporting ranking and clustering as generalized order-by and group-by. In *SIGMOD*, 2007.

[14] B. Liu et al. Clustering through decision tree construction. In *CIKM*, 2000.

[15] U. Manber et al. Experience with personalization of Yahoo! *Commun. ACM*, 43(8), 2000.

[16] H. S. Nagesh. High performance subspace clustering for massive data sets. In *Master's thesis*. 1999.

[17] L. Parsons et al. Subspace clustering for high dimensional data: a review. *SIGKDD Explorations*, 6(1), 2004.

[18] K. A. Ross and A. Janevski. Querying faceted databases. In *SWDB*, 2004.

[19] S. B. Roy et al. Minimum effort driven dynamic faceted search in structured databases. In *CIKM*, 2008.

[20] Y. Sun et al. RankClus: integrating clustering with ranking for heterogeneous information network analysis. In *EDBT*, 2009.

[21] B. Wynar. *Introduction to Cataloging and Classification*. Libraries Unlimited, 1992.