Preprint of:
K. Manioudakis and Y. Tzitzikas, Extending Faceted Search with Automated Object Ranking,
13th Metadata and Semantics Research Conference (MTSR 2019), Rome, Italy, Oct 2019

Extending Faceted Search with Automated Object Ranking

Kostas Manioudakis [0000-0002-6510-9889] and Yannis Tzitzikas [0000-0001-8847-2130]

Institute of Computer Science, FORTH-ICS, Heraklion, Greece, and
Computer Science Department, University of Crete,
{manioudaki, tzitzik}@ics.forth.gr

Abstract. Faceted Search is a widely used interaction scheme in digital libraries, e-commerce, and recently also in Linked Data. Nevertheless, object ranking in the context of Faceted Search is not well studied. In this paper we propose an extended version of the model enriched with parameters that enable specifying the characteristics of the sought object ranking. Then we provide an algorithm for producing an object ranking that satisfies these parameters. For doing so various sources are exploited including preferences and statistical properties of the dataset. Finally we present an implementation of the model, the GUI extensions that were required, as well as simulation-based evaluation results that provide evidence about the reduction of the user's cost.

1 Introduction

Faceted Search (FS) is the de facto query paradigm in e-commerce for more than one decade [14,16]. It is widely used in digital libraries, in the semantic web and in Linked Data. FS is essentially a *session-based* interactive method for *gradual query formulation* (commonly over a multidimensional information space) through *simple clicks* that offers to the user an *overview* of the result set (groups and count information) and *never leads to empty result sets*. At each state of the interaction the user explores the *focus*, i.e. the set of objects that satisfy the various constraints/filters that the user has specified up to that point. These objects are *unranked*, e.g. when the user explores a catalogue for buying a new laptop, or *ranked*, e.g. when the user explores the available hotels which are ordered with respect to price, user ratings or other criteria (default or specified by the user in the form of preferences as in the case of PFS [19]). The focus is ranked also in cases FS is applied after a keyword search query, e.g. as in Google Scholar. Although object ranking in FS is already used in commercial systems, the scientific literature on this topic is relatively short. Most of the research, has focused on methods only for facet ranking, i.e. for deciding in what order to place the facets. In this paper we focus on the *ranking of objects*. We propose an extension of the FS model that is enriched with parameters for specifying the desired properties of object ranking. These parameters allow tackling the problem of too big or too small answers and can specify how refined the sought ranking should be, as well as how long the answer should be. Then we describe

methods and algorithms for deriving such object rankings. In the sequent we describe how we have realized the model. To grasp the idea, the left side of Figure 1 sketches the GUI of a typical FS containing three facets and a focus comprising 8 objects partitioned into 2 buckets, the first containing 6 objects and the second 2 objects. The right side sketches the GUI according to the extended model that we introduce, where the user has asked for 10 objects, and a more refined ranking, specifically that no bucket should contain more that 3 objects. We can see that more objects have appeared (approximate results) and the focus respects the Maximum Block (MB) constraint of 3.

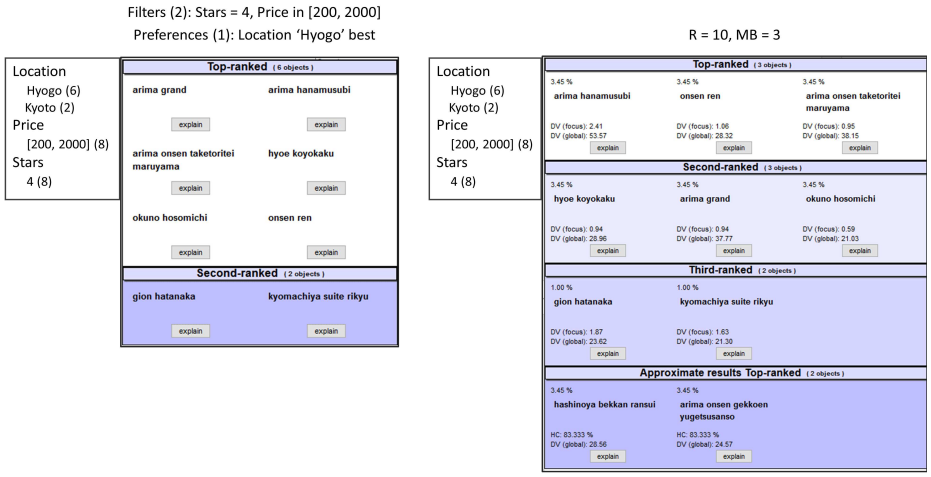

Fig. 1: Impact of automatic ranking on search results. The approximate results satisfy the preference and one of the 2 filters.

In a nutshell, the main contributions of this paper are: (a) the extension of the model with object ranking-related parameters, (b) the formulation of the corresponding object ranking problem, (c) the discussion on the ability to solve the problem, (d) the algorithm `SmartFSRank` for providing a solution, (e) the proposal of how to extend the GUI of FS systems for making evident and clear the object ranking, (f) the description of the model's implementation, and (g) some promising simulation-based evaluation results. The rest of this paper is organized as follows: Section 2 describes the context and related work. Section 3 introduces the extended model. Section 4 provides the algorithms for realizing the extended model. Section 5 describes the extensions of the GUI and the implementation, Section 6 discusses related systems and some evaluation results. Finally, Section 7 concludes the paper.

2 Context and Related Work

2.1 Background

Faceted Search *Faceted Exploration* (or *Faceted Search*) is a widely used interaction scheme for Exploratory Search. It is the de facto query paradigm in

e-commerce [14,16]. It is also used for exploring RDF Data (e.g. see [18] for a recent survey, and [10] for a recent system). In a short (and rather informal) way we could define it as a *session-based interactive method for query formulation (commonly over a multidimensional information space) through simple clicks that offers an overview of the result set (groups and count information), never leading to empty result sets* .

PFS: Preference-enriched Faceted Search *Preference-enriched Faceted* Search [19], for short PFS, is an extension of FS that supports *preferences*. PFS offers actions that allow the user to order facets, values, and objects using *best, worst, preferTo* actions (i.e. relative preferences), *aroundTo* actions (over a specific value), and other criteria. Furthermore, the user is able to *compose* object related preference actions, using *Priority, Pareto, Pareto Optimal* (i.e. skyline) and other. The distinctive features of PFS is that it allows expressing preferences over attributes whose values can be hierarchically organized (and/or multi-valued), it supports preference inheritance, and it offers scope-based rules for resolving automatically the conflicts that may arise. As a result the user can restrict his focus by using the faceted interaction scheme (hard constraints) that lead to non-empty results, and rank according to preference the objects of his focus. PFS has been used in various domains, also in the context of spoken dialogue systems [12]. Recent extensions of SPARQL with preferences ([13,15]) will facilitate the implementation of PFS over RDF data.

2.2 Related Work

There is related work from several areas including FS systems, Databases, and Learning to Rank approaches.

In Faceted Search Systems. The idea of automatic ranking in faceted search is not new, and there are various approaches for automatic ranking in faceted search systems, discussed in the survey [18]. Most of the research, has focused on methods only for *facet ranking*, i.e. for deciding in what order to place the facets. In most cases (e.g [8], [7]), the proposed methods are based on the frequency of facet values. The approach in [6] dynamically ranks the properties depending on the query. Moreover, they define different measures on qualitative and numeric facets, and also suggest ranking the values of each property by their frequency in descending order. Other metrics have also been utilized such as navigational cost [5]. In [3] a set-cover ranking method is used. Ranking the facets can be also useful for ordering the objects.

Object ranking, is already used in commercial systems. For instance, the hotel platform booking.com offers a ranking method based on various properties. However, the scientific literature on this topic is relatively short. In [14] (Chapter 9) a method based on property values is briefly described through an e-shopping example. According to PFS, and the system Hippalus [11], the user can specify the ranking of objects through preference actions. In the context of PFS, [17] introduced a method for quantifying the degree of match between an object and the user's preference actions. This is mainly for making evident to the user how

good the top-ranked objects are. In the same context of PFS, [12] introduces selectivity and entropy, that can be used for ranking the objects.

In Databases. The problem of automated ranking has also been studied in the context of databases. For instance, [1] aims at tackling the Many Answers Problem, which arises when a query returns too many result tuples without any ordering. That work proposes a framework that adopts the popular Information Retrieval ranking method of cosine similarity on TF-IDF weights. An alternative approach for the Many Answers Problem was proposed in [2] that is based on Probabilistic Information Retrieval. In that work, they investigate attributes not specified in the query and calculate two scores: a *global score* which captures the global importance of unspecified attribute values and a *conditional score* which captures the strengths of dependencies (or correlations) between specified and unspecified attribute values. These scores are estimated automatically from a workload of *past queries* and with data analysis. A user evaluation showed better quality in comparison to methods of [1].

Learning to Rank Approaches. In the field of Information Retrieval apart from the classical retrieval models, lately various methods that utilize Machine Learning have been developed [9]. These processes involve creating a ranking function by training a machine learning model on user data, collected from past queries (e.g. number of clicks on each item, explicit user ratings, etc). A Learning to Rank approach for faceted search is proposed in [20]. In that work, a weighted TF-IDF scoring formula calculated on facets is adopted. Queries and documents are represented as sets of facet-value pairs, and learning methods are employed to determine the optimal weights of the formula, given a user query and a set of previously judged documents (from explicit and implicit user feedback).

3 The Extended Model

Representation of Data. We assume a set of objects Obj described by a set of facets $F_1, \ldots, F_k$. The values of these facets can be categorical, numerical, as well as values from a taxonomy. Moreover facets can be multi-valued. Overall this representation framework captures to most widely (and successful) applications of FS (for e-commerce, booking application, bibliographic search, etc).

Plain FS Interaction. The user during the FS formulates interactively a set of hard constraints (filters), denoted by hc. The focus is actually the extension of the hc, denoted by $E(hc)$ ($E(hc) \subseteq Obj$).

FS with Preferences. If PFS is supported and the user has expressed a set of preferences, i.e. a set of *soft constraints* denoted by sc, then they are exploited for ranking the elements of $E(hc)$ with respect to the preference relation $\succ_{sc}$.

The Focus. In general we can consider that the focus at each state of the interaction is a *bucket order*, $L = \langle b_1, \ldots, b_z \rangle$, i.e. a sequence of blocks $b_1 \ldots b_z$, each being a subset of Obj which are pairwise disjoint ($b_i \cap b_j = \emptyset$). Note that z could be 1, meaning that the focus consists of a single block with objects not ranked (or ranked arbitrarily for reasons of presentation). Note that this formulation captures both FS and PFS, which uses preferences to rank the objects.

User Session. A user session us is a series of actions, $s = \langle a_1, \ldots, a_n \rangle$ where each a_i is a hard or soft constraint.

Parameters of the Extended Model. We propose extending the model with two parameters

- MB: Maximum Block size. E.g. if $MB = 1$ then the system should return a linear order of objects, if $MB = 2$ the answer should not contain ties between more than 2 objects.
- R: number of requested objects.

With these two parameters several requirements can be tackled:

- Too many objects: R forces the system to rank the objects for returning the best R objects.
- Too few objects: R forces the system to also return approximate objects
- Arbitrary order: MB forces the system to rank the objects so that no block has more than MB objects, and in this way the rank is not arbitrary

Characterizing a bucket order L. Let L be a bucket order $L = \langle b_1, \ldots, b_z \rangle$, i.e. a possible ranked answer, and let $objects(L)$ denote the set of objects that occur in L. We could characterize L according to various criteria:

- **HCsat**. We could say that L satisfies the hard constraints hc, if $objects(L)$ are exactly those that satisfy hc, i.e. $objects(L) = E(hc)$.
- **SCsat**. We could say that L satisfies a soft constraint sc (i.e. it respects the preference order), if $L \supseteq \succ_{sc|E(hc)}$. This means that autoranking is used only for ranking the objects in each block of the preference order (it never "violates" the blocks, it just add relationships, and these relationships do not create any cycle).
- **MBsat.** We could say that L satisfies a *maximum allowable block size* MB, if $|b_i| \leq MB$ for each $1 \leq i \leq z$.
- **Rsat.** We could say that L satisfies R, if L contains exactly R objects.

It is not hard to see that it is not always possible to find an L that satisfies all of the above constraints. For instance if the objects that satisfy the hc are less than R, i.e. $|E(hc)| < R$, then the system should either return less objects (sacrificing Rsat), or should try to return R objects by extending $E(hc)$ with the $R - |E(hc)|$ in number "closest" objects (sacrificing HCsat).

On the other extreme, if those that satisfy the hc are more than R, then the system should rank them and return the best of them. This is another case of an L that is not HCsat, since it contains less objects than those satisfying hc. A problem statement that includes more requirements follows:

Definition 1 (Problem Statement). Given a user session us with hard and soft constraints, a parameter R specifying the number of desired objects, a parameter MB specifying the maximum allowable block size, compute and return to the user the "best" (wrt HCsat, SCsat, MBSat, Rsat) answer L. □

What remains is to clarify what "best" means. A first objective is to produce one or more L that satisfy all the criteria if possible. In case there are more than one, then we need a method to select one of them. To this end, dataset statistics and other metrics (as we have seen in the related work) can be used. For example we could promote frequently or rarely occurring values. If there is

no L that satisfies all hard constraints, then we need to find one that "better approximates" a bucket order that satisfies them all. We will elaborate on this issue in the next section.

4 The `SmartFSRank` Ranking Method

Here we define an algorithm that provides a solution to the problem statement (as defined in Def. 1). In general the algorithm exploits PFS, if supported, and it tries to satisfy R by ranking and approximate matching, and MB through ranking based on statistical properties of the data. Note that the algorithm can be applied even if PFS is not supported (in that case we have one block, i.e. $z = 1$). In brief, the algorithm `SmartFSRank`, Alg. 1, first tries to satisfy hc (line 1), and then sc (Part 1, line 2) by exploiting the PFS-based ranking method. Then it tries to satisfy R (Part 2, lines 5-7), and finally MB (Part 3, lines 8-13). In Part 2, if R is greater than the size of the current focus, then more objects (not satisfying hc) have to be added and this should be based on the approximate satisfaction of the hard constraints. This selection is done by **AppendBlocks** that is analyzed in 4.1. In Part 3, the block breaking for satisfying MB can be based on frequency, and similarity to soft constraints, as defined in [17], and it is done by **BreakBlock(b)** that is analyzed in 4.2.

Algorithm 1 SmartRank
Input: Obj, hc, sc, MB, R
Output: A ranked answer that satisfies hc, sc, MB and R.

1: $A = E(hc)$; ▷ The objects satisfying the hc
2: */** Part (1): Apply the PFS method to satisfy sc*/*
3: Compute $(A, \succ_{sc\ |A})$ which is a series of blocks $= \langle b_1, \ldots, b_Z \rangle$.
4: */** Part (2): Satisfy R */*
5: **if** $|A| < R$ **then** ▷ need to add more objects
6: $Result = Result$.**AppendBlocks**$(R - |A|)$; ▷ Add new blocks to the answer
7: **end if**
8: */** Part (3): Satisfy MB*/*
9: **for** each $b \in Result$ **do**
10: **if** $|b| > MB$ **then** ▷ If block b does not satisfy MB
11: Replace b by **BreakBlock**(b, MB, sc, hc)
12: **end if**
13: **end for**

4.1 AppendBlocks

The idea is to score each object according to its distance from the hc. Having such a scoring function, a method to realize AppendBlocks is to score each object not in $E(hc)$ and return the $R - |E(hc)|$ objects that have the highest score. This method guarantees that it will return the objects which maximize the score, i.e. those that better approximate the information need, as expressed

by the *hc*. As regards the scoring, below we detail a method based on facet types. Let $hc = c_1 \wedge \ldots \wedge c_n$, where each c_i is a constraint like $F_i = t_i$, e.g Stars = 4. If o_j is an object, with o_{ji} we denote the value of o_j on the facet F_i. In Table 1 we define the score per conjunct based on the facet type. The first column corresponds to the case where a conjunct is satisfied, while the second presents the formulas used when a conjunct is not satisfied. In both cases the scores are in the interval [0, 1]. In the last row, corresponding to the case where the terminology is structured as a taxonomy, we define a similarity measure that reflects the distance in the taxonomy. For a term $x \in F_i$, let $up(x) = \{\ t \in F_i \mid x \leq_i t\}$. We define the similarity between two terms x and y, as the Jaccard similarity of their greater nodes, specifically: $sim_{tax}(x, y) = \frac{|up(x) \cap up(y)|}{|up(x) \cup up(y)|}$ and then define $score_{taxonomy}(F_i = t_i, o_j) = sim_{tax}(t_i, o_{ji})$.

Table 1: Formulas for calculating $score_{hc}$ per conjunct

score type	o_j satisfies c_i	o_j does not satisfy c_i
$score_{flatterminology}(F_i = t_i,\ o_j)$	1	0
$score_{numeric}(F_i = t_i,\ o_j)$	1	$1 - \frac{\lvert t_i - o_{ji}\rvert}{\max_{o_m \in Obj}\{\lvert t_i - o_{mi}\rvert\}}$
$score_{interval}(F_i \in [a, b],\ o_j)$	1	$score_{numeric}(F_i = \frac{a+b}{2},\ o_j)$
$score_{taxonomy}(F_i = t_i,\ o_j)$	$sim_{tax}(t_i, o_{ji})$	$sim_{tax}(t_i, o_{ji})$

Back to the running example of Fig. 1, we can now see how the scores of the approximate results were calculated, regarding the constraint Stars = 4. They both have Stars = 5, so their score [1] according to the numeric case of the above table is: $score_{numeric}(Stars = 4,\ 5) = 1 - \frac{|4-5|}{4-0} = 1 - \frac{1}{4} = 0.75$

Definition 2 (HCscore). We can define the consolidated score of an object o with respect to $hc = c_1 \wedge \ldots \wedge c_n$, as: $score_{hc}(o_j) = \frac{1}{n} \sum_{i=1}^{n} score_{type(c_i)}(o_j)$ where $type(c_i) \in \{flatterminology, numeric, interval, taxonomy\}$.

This formula can be considered as the *baseline*. It expresses how close o_j is with respect to a point that satisfies *hc*. The exact algorithm for AppendBlocks is Alg. 2. It returns the bucket order to be appended.

4.2 BreakBlock

For breaking each block that does not satisfy MB an idea is to score each object of the block according to its *discrimination value*. Rare elements are harder to find, therefore it could be reasonable to promote objects that have rare values, i.e. those with higher discrimination value. On the other hand, frequent values may correspond to popular values, therefore it could be also reasonable to promote frequent values, i.e. those that appear in several objects. As we shall see in the section about GUI, the GUI allows the user to specify whether rare or frequent values are preferred. In any case we have to define and compute the discrimination value. Having such a formula we can apply it to each object of

[1] Assuming that the range of the facet Stars, in the dataset, is the set {0,1,2,3,4,5}.

Algorithm 2 AppendBlocks
Input: Num, hc
Output: A bucket order to be appended

```
1: CO = Obj \ E(hc);                                   ▷ The candidate objects
2: for each o ∈ CO do
3:     o.score = score_hc(o);                          ▷ compute the score of o wrt hc
4: end for
5: Sort(CO, score, descending);          ▷ sort CO wrt score attribute in desc. order
6: Let B = ⟨b_1, ..., b_F⟩ the resulting blocks          ▷ after the previous sorting
7: c = 0;  i = 1;  A = ϵ;
8: while c < Num do          ▷ While the objective of Num objects is not reached
9:     A = A.appendBlock(b_i);  i++;  c = c + |b_i|;
10: end while
11: Return A;
```

any block that does not satisfy MB to order its objects. Then, such blocks will break to smaller ones satisfying MB. We can define the *discrimination value* (dV) of a tuple $\boldsymbol{o_j} = (o_{j1}, \ldots, o_{jk})$ by taking the average inverse frequency, i.e.: $dV_w(\boldsymbol{o_j}) = \frac{1}{k} * \sum_{i=1,k} \frac{1}{freq_w(o_{ji})}$. Note that frequency can be defined in various ways, this is why the above formula uses $freq_w$ where $w \in \{g, ga, E\}$. Specifically the frequency of a value $t \in F_i$, can be defined *globally* ($freq_g$), or with respect to the objects that *have value in facet* F_i ($freq_{ga}$), or in the *current focus* E ($freq_E$). Formally: $freq_g(o_{ji}) = \frac{|\{o_x \in Obj | o_{xi} = o_{ji}\}|}{|Obj|}$ (1),
$freq_{ga}(o_{ji}) = \frac{|\{o_x \in Obj | o_{xi} = o_{ji}\}|}{|\{o_y \in Obj | o_{yi} \neq \epsilon\}|}$ (2), $freq_E(o_{ji}) = \frac{|\{o_x \in E | o_{xi} = o_{ji}\}|}{|E|}$ (3).
Note that if $o_{ji} = \epsilon$, i.e. null, then $freq_{\{g,E\}}(o_{ji})$ = number of objects having null (just like an ordinary value). In general it makes sense to consider a *series of "tie breaking" methods*, for making sure that all ties can be broken so that MB is eventually satisfied. Each such method can be assigned a level, meaning that if the application of the level i method does not break a tie, then the level $i+1$ method is applied. The exact algorithm for BreakBlock is Alg. 3.

It first breaks the block b with respect to the discrimination value in the focus (i.e. $freq_E$). If MB is still not satisfied, it uses $freq_g$ (recursively only for that block). Specifically we assume the following series of levels: $Levels = \langle dv_E, dv_G, lexicographic \rangle$. At level 3 we assume the lexicographic order wrt the name of the object. This ensures that the algorithm terminates. The algorithm returns a bucket order that certainly satisfies MB. One key point is that a cost is paid only if needed i.e. only for the blocks that do not satisfy MB.

5 Extensions of the Graphical User Interface

In general we have identified the following GUI issues: (a) how to make evident the automatic ranking, (b) how to enable the user to change the ranking (e.g. frequent vs rare), (c) how to make clear the objects that do not satisfy the hard constraints, (d) how to provide ranking explanation (both for *hc* and

Algorithm 3 BreakBlock
Input: $b, MB, level, hc, sc$, where b does not satisfy MB.
Output: A bucket order of the objects of b that satisfies MB.

1: $A = objects(b)$; ▷ the objects occuring in b
2: **for** each $o \in A$ **do**
3: $o.dv = DV(o, level)$; ▷ compute the discrimination value of o at $level$
4: **end for**
5: $Sort(A, dv, descending)$; ▷ sort A wrt dv attribute in desc. order
6: Let $B = \langle b_1, \ldots, b_F \rangle$ the resulting blocks ▷ after the previous sorting
7: **for** each $b_i \in B$ **do**
8: **if** $|b_i| > MB$ **then** ▷ if b_i still does not satisfy MB
9: $Bnew =$ **BreakBlock**$(b_i, MB, level + 1, sc, hc)$; ▷ recursive call with +1 level
10: **end if**
11: Replace in B the block b_i by the series of blocks $Bnew$
12: **end for**
13: Return B;

sc). For the implementation of the extended FS model that we propose in this paper, we decided to use `Hippalus` which is a publicly accessible web system that implements the PFS interaction model. The information base that feeds `Hippalus` is represented in RDF/S[2] using a schema adequate for representing objects described according to dimensions with hierarchically organized values. Below we describe how we tackled the above questions by showing screenshots from our implementation. The bucket order is presented by separating buckets with a line label "Top-ranked", "Second-ranked", etc. so that the preference-based ranking is made clear to the user. The objects within a preference-based bucket are ordered based on the automatic method presented in this paper (instead of an arbitrary one). The settings provided are following (see also Fig. 2):

1. Enable / disable Rsat
2. Specify the value of R parameter
3. Enable / disable inner bucket ordering
4. Enable / disable MBsat
5. Specify the value of MB parameter
6. Select policy about discrimination value: prefer rare values, common values or no preference

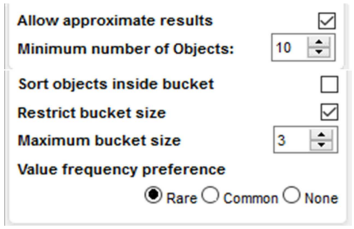

Fig. 2: The automatic ranking settings as provided in the GUI

Figure 2 displays the settings used in the running example (Fig. 1). As regards *ranking explanation*, (a) the GUI shows the scores for each object (consolidated

[2] http://www.w3.org/TR/rdf-schema/

$score_{hc}, dv_E, dv_G$, and similarity to soft constraints, as a percentage) as shown in the right side of Fig. 1, and (b) the GUI provides a button labeled "explain" on each object, that when pressed, it displays the $score_{hc}$ per facet, as well as the soft constraints and which of them are satisfied by the object.

6 Discussion

Comparison with Related Systems. In comparison to the existing work, at first we should say that the data space we assume supports hierarchically organized values. As regards the proposed ranking framework, it is the first that considers hard constraints (the typical functionality of FS), soft constraints (including preference inheritance in the hierarchically organized values), as well as statistical-based object ranking. It does not require log or training data, therefore it can be widely applied. Probably the closest work to ours is [20] however they rely on machine learning techniques and user data, while we focus on the query constraints and the statistical properties of the dataset. To make clear the key differences between our system and the most related systems that were mentioned in §2, Table 2 provides a list of features and marks those systems that provide them.

Table 2: Comparison with Related Systems

	Our Tool	GRAFA [10]	Basu et al. [4]	van Belle [20]
Facet Hierarchies	Yes	No	Yes	No
Approximate Matching	Yes	No	No	Yes
Blocks of desired size	Yes	No	No	No
Preferences	Yes	No	No	No
Needs training data	No	No	No	Yes

Simulation-based Evaluation. The main purpose of automatic ranking, is to assist the user in finding the desired object. One way to measure this gain, is to consider the number of constraints required, until the desired objects is ranked in the top-K positions of the focus (for various values of K). The better the automatic ranking is, the less constraints are required. To this end, we have started a simulation based evaluation, since this can be more objective and less laborious than evaluation with users. So far we have made experiments using one dataset, which contains information about 382 hotels and has 18 facets. We considered each of these 382 hotels as a possible ideal top result. We run 2 tests. In the first one, we simulated users that try to find that hotel by sequentially applying hard constraints that match the object's description (the conjuncts correspond to the object's facet-value pairs). In the second one, instead of hard constraints the user formulated only soft constraints. Specifically, for each facet-value pair of the target object a soft constraint of type *"best"* was applied (e.g *prefer Stars...4 best*). The initial ordering of the objects was a random one, but the same in each simulated session. Below we report the average and maximum number of constraints needed in each of three cases: (1) No automatic ranking, (2) Automatic ranking based on discrimination value, preferring rare values, (3) Automatic ranking based on discrimination value, preferring common values.

For cases (2) and (3) the maximum block size (MB) is 1, so that there aren't any ties in the ranking. Tables 3 and 4 show the simulation results.

Table 3: Hard constraints session simulation results

Automatic Ranking	Preference on DV	Avg. HC (K=1)	Max HC (K=1)	Avg. HC (K=2)	Max HC (K=2)	Avg. HC (K=3)	Max HC (K=3)
Disabled	-	4.55	19	3.08	15	2.51	14
Enabled	Rare	4.85	19	3.29	18	2.62	15
Enabled	Common	**4.11**	**18**	**2.92**	**10**	**2.45**	**10**

Table 4: Soft constraints session simulation results

Automatic Ranking	Preference on DV	Avg. SC (K=1)	Max SC (K=1)	Avg. SC (K=2)	Max SC (K=2)	Avg. SC (K=3)	Max SC (K=3)
Disabled	-	4.26	16	2.99	14	2.46	14
Enabled	Rare	4.43	16	3.14	14	2.57	14
Enabled	Common	**3.99**	**14**	**2.89**	**11**	**2.43**	**7**

Although we plan to continue evaluation on more datasets and with more complex simulated user sessions, the results so far show that preferring *common* values is better than searching without automatic ranking. This holds for all metrics that are measured in Tables 3 and 4. On the contrary, if preference is given on rare values, we get worse results than those with arbitrary ranking. The above provide positive evidence for the benefits of MB and of BreakBlocks, specifically they reduce the average cost up to to 9.18% (HC, K=1) and the maximum cost up to 50% (SC, K=3). As regards the benefits from R, it is not hard to see that whenever a new approximate object is added, it reduces the number of constraints the user would have to formulate (for getting that object). Therefore the gain from adding $R-|A|$ objects is the number of distinct descriptions of these objects, so the gain ranges $[1, R-|A|]$.

Efficiency. Although scalability is not currently our focus, we should note that the time complexity of the ranking methods is $O(N * K)$ where N, K are the number of objects and facets. For the dataset we used, which contained 382 objects, the average time to apply a hard constraint increased from 2 ms (without automated ranking) to approximately 20 ms (with automated ranking), which is not noticeable by the user.

7 Concluding Remarks

We proposed an extended model for FS for improving the exploration experience of the users in various contexts. Specifically, we proposed a set of parameters for specifying the desired properties of *object ranking*, and then, through `SmartFSRank`, we factorized the problem to sub-tasks that can be tackled more easily. Finally, we showed the model's implementation and the required GUI extensions. The evaluation results of the extended model through simulation are promising, i.e. they provide evidence that the proposed model reduces the user cost for finding the desired object, specifically it reduces the average cost up to 9.18% and the maximum cost up to 50%. Apart from continuing the simulation-based evaluation, an interesting extension would be to consider diversification

requirements, as well as to investigate indexes and algorithms for scalability i.e. for enabling FS with automated ranking over very big datasets.

Acknowledgement. This work was partially supported by the project AI4EU (EU H2020, Grant agreement No 825619).

References

1. S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated ranking of database query results. In *CIDR*, 2003.
2. S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic ranking of database query results. In *Proceedings of the Thirtieth VLDB*, 2004.
3. W. Dakka, P. Ipeirotis, and K. Wood. Automatic construction of multifaceted browsing interfaces. In *Proceedings of the 14th CIKM*, 2005.
4. Basu et al. Minimum-effort driven dynamic faceted search in structured databases. In *Proceedings of the 17th CIKM*. ACM, 2008.
5. Li Chengkai et al. Facetedpedia: Dynamic generation of query-dependent faceted interfaces for wikipedia. In *Proceedings of the 19th ICWWW*. ACM, 2010.
6. Vandic et al. Dynamic facet ordering for faceted product search engines. *IEEE Transactions on Knowledge and Data Engineering*, 2017.
7. R. Hahn, C. Bizer, C. Sahnwaldt, C. Herta, S. Robinson, M. Bürgle, H. Düwiger, and U. Scheel. Faceted wikipedia search. In *Business Information Systems*, 2010.
8. Andreas Harth. Visinav: Visual web data search and navigation. In *Database and Expert Systems Applications*, 2009.
9. Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3, 2009.
10. J. Moreno-Vega and A. Hogan. Grafa: Scalable faceted browsing for RDF graphs. In *ISWC*, 2018.
11. P. Papadakos and Y. Tzitzikas. Hippalus: Preference-enriched faceted exploration. In *EDBT/ICDT Workshops*, volume 172, 2014.
12. A. Papangelis, P. Papadakos, Y Stylianou, and Y. Tzitzikas. Spoken dialogue for information navigation. In *SIGDial*, 2018.
13. O. Pivert, O. Slama, and V. Thion. SPARQL Extensions with Preferences: a Survey. In *ACM Symposium on Applied Computing*, 2016.
14. G. M. Sacco and Y. Tzitzikas. *Dynamic taxonomies and faceted search: theory, practice, and experience.* Springer Science & Business Media, 2009.
15. A. Troumpoukis, S. Konstantopoulos, and A. Charalambidis. An extension of sparql for expressing qualitative preferences. In *ISWC*, 2017.
16. D. Tunkelang. Faceted search. *Synthesis lectures on information concepts, retrieval, and services*, 2009.
17. Y. Tzitzikas and E. Dimitrakis. Preference-enriched faceted search for voting aid applications. *IEEE Transactions on Emerging Topics in Computing*, 7(2), 2019.
18. Y. Tzitzikas, N. Manolis, and P. Papadakos. "Faceted Exploration of RDF/S Datasets: A Survey". *Journal of Intelligent Information Systems*, 2017.
19. Y. Tzitzikas and P. Papadakos. Interactive exploration of multidimensional and hierarchical information spaces with real-time preference elicitation. *Fundamenta Informaticae*, 20:1–42, 2012.
20. Agnes van Belle. Learning to rank for faceted search: Bridging the gap between theory and practice, 2017.