A Framework for Approximate Product Search Using Faceted Navigation and User Preference Ranking

Damir Vandic[a], Lennart J. Nederstigt[b], Flavius Frasincar[a], Uzay Kaymak[c], Enzo Ido[a,*]

[a]*Erasmus University Rotterdam, PO Box 1738, 3000 DR, Rotterdam, the Netherlands*
[b]*Grible.co, Coolsingel 104, 3011 AG Rotterdam, the Netherlands*
[c]*Eindhoven University of Technology, PO Box 513, 5600 MB, Eindhoven, the Netherlands*

Abstract

One of the problems that e-commerce users face is that the desired products are sometimes not available and Web shops fail to provide similar products due to their exclusive reliance on Boolean faceted search. User preferences are also often not taken into account. In order to address these problems, we present a novel framework specifically geared towards approximate faceted search within the product catalog of a Web shop. It is based on adaptations to the p-norm extended Boolean model, to account for the domain-specific characteristics of faceted search in an e-commerce environment. These e-commerce specific characteristics are, for example, the use of quantitative properties and the presence of user preferences. Our approach explores the concept of facet similarity functions in order to better match products to queries. In addition, the user preferences are used to assign importance weights to the query terms. Using a large-scale experimental setup based on real-world data, we conclude that the proposed algorithm outperforms the considered benchmark algorithms. Last, we have performed a user-based study in which we found that users who use our approach find more relevant products with less effort.

Keywords: approximate product search, faceted navigation, user preferences

1. Introduction

E-commerce has nowadays become very popular among consumers. However, there are many Web shops with very large product catalogs, which makes it difficult for users to find their desired product. It has been shown that many users encounter this difficulty while shopping online [1, 2],

*Corresponding author; tel: +31 (0)10 408 1262
Email addresses: vandic@ese.eur.nl (Damir Vandic), nederstigt@grible.co (Lennart J. Nederstigt), frasincar@ese.eur.nl (Flavius Frasincar), u.kaymak@tue.nl (Uzay Kaymak), 534787ei@eur.nl (Enzo Ido)

which can negatively impact the turnover for Web shop owners. Studies from the past have shown that factors other than the price play a crucial role when a consumer decides to choose where to buy a product online [3]. This emphasizes the need for better search interfaces for browsing through product catalogs on the Web.

Throughout the years various interaction paradigms have been proposed to deal with the above problem. Faceted search is one of the most popular paradigms for browsing structured data in information systems [4, 5], especially product catalogs in e-commerce stores [6, 7]. One of the reasons why faceted search is popular among Web shops is that users find it intuitive [8, 9]. This paradigm is useful for exploratory search, as users can iteratively select facets that they find important to constantly refine their query and enhance the result set [10, 9, 11]. Furthermore, it has been shown that faceted search is perceived as easy to understand and effective at guiding people to relevant documents within catalogs [11].

Despite the advantages of faceted search, there are also some serious drawbacks. Traditionally, faceted search imposes a rather strict Boolean model on whether a document matches the query terms: either it matches the query or it does not. While this model allows for a quick drill-down into the catalog, it also means that a user might miss some potentially desired products if they do not completely match the query. This results in a user having to change the query afterwards to include more documents, which increases search time and makes exploratory search more error-prone. Such strict faceted search is not desired in e-commerce, as users usually do not have completely strict requirements.

Most of the currently proposed approaches that aim to tackle this problem are not specifically geared towards the domain of e-commerce. In particular, the existence of quantitative facets, such as price, might pose a problem to these algorithms, as they do not consider nearly identical numerical values as being similar to a certain degree. Furthermore, past approaches for e-commerce also do not allow the user to explicitly specify which requirements are important and which ones are not. To our knowledge, a faceted search method that unifies the ranking of products and the ranking of user requirements has not yet been proposed.

In this paper, we propose a novel algorithm specifically geared towards approximate faceted search within a product catalog of a Web shop. The main focus is on improving the retrieval of relevant products, taking into account the importance of the specified user requirements. We adapt

the p-norm extended Boolean model [12] to cope with the domain-specific characteristics of faceted product search. The adaptations consist of a facet similarity function, both for quantitative and qualitative product properties. We also propose a term weighting method that takes in account user preferences. The main advantage of the proposed method is that it is purely data-driven and does not rely on external knowledge. The algorithm is implemented as part of a Web application, with real-life data obtained from Tweakers.net Pricewatch [13], which is a Dutch Web shop price aggregation service. An extensive evaluation is performed by means of simulations and a battery of quality and performance metrics.

The contributions of this paper are threefold. First, we propose an approximate search approach that is specifically geared towards e-commerce, where faceted navigation is a commonly used user interface paradigm. Second, our approach extends the functionality of approximate search by allowing users to specify the importance of the selected product facets. Third and last, the approach that we propose introduces minimal changes to the classical faceted navigation systems present in current Web shops, making our solution a contribution to practice as well.

2. Related Work

In the literature, we can find some proposed approaches that aim to improve exploratory search within a product catalog using recommender systems [14, 15].

The author of [15] proposes a recommender system based on case-based reasoning. This system matches a user's need with a product based on previously solved user needs that are similar to the current user's needs. The authors augment this approach by combining it with example critiquing, which shows a product to the user and then allows the user express his/her dissatisfaction with a particular facet of that product. This information can then be used to find another product that has a better value for this facet, possibly at the cost of having a worse value for other facets, thereby facilitating trade-off-based navigation.

Similarly, in [16] an approach is proposed that uses example critiquing techniques. They propose an interaction paradigm in which users first declare their initial preferences, after which some matching example documents are displayed. The user subsequently examines these examples and either accepts one document or performs critiquing on one of the examples. This process consists of improving some of the attribute values by allowing a compromise on one or more other attribute

values. Afterwards, the system applies a decision strategy called the weighted additive sum rule, which multiplies each attribute value of a document with the corresponding importance weight in the query, followed by adding up these values to obtain an overall score. The documents with the highest scores are displayed to the user, after which the user can either continue to perform exploratory search or accept one of the presented documents. The authors show that users were able to improve their decision accuracy by using the trade-off support provided by this system.

In [17], the authors examine an approach that employs concepts from the field of utility theory. They argue that buying a product is different from finding relevant products, therefore it is possible to calculate the expected utility of products and rank them based on the largest utility surplus. The utility surplus is defined as the difference between two components, namely the utility of the product and the utility of money. The utility of the product is defined as the sum of expected utilities for each of its characteristics, whereas the utility of money is defined as an increasing concave function. In order to obtain the expected utilities, they collect aggregated data about market shares and demand, from which they derive an estimated utility. They show that users tend to prefer their surplus-based ranking when searching for hotels in comparison to other considered rankings.

Although faceted search is regarded as a useful means to explore a catalogue, care needs to be taken that the interface does not become too cluttered and obscured, which could result in a diminished user performance due to information overload [18]. General concepts from the literature for implementing faceted navigation are explained and reviewed in [19]. One of the ideas discussed for reducing the overload of information is to either filter or order the facets according to their importance.

Displaying all facets may be a solution when a small number of facets is involved, but it can overwhelm the user for larger sets of facets [18, 20]. Studies such as [21, 22, 23, 24, 25, 26, 27, 28, 29] focus primarily on optimizing the decision of which facets to show in the user interface. These algorithms aim to reduce user effort by ordering the facets in such a way that the most important ones are listed first.

The study in [27] is based on earlier work of the authors which coined a basic idea for selecting more descriptive facets using an entropy-based measure [30]. The approach focuses on Semantic Web and Linked Data data sets, similar to the approaches in [31, 32, 33, 34] The reduction in user

effort is achieved by calculating gain ratios for each facet, which is based on conditional entropy. The conditional entropy indicates how much the value for a facet can be predicted by the value for another facet. By summing up all the gain ratios of the facets belonging to each property, the total gain ratio per property is computed. Last, the properties are ranked in decreasing order of total gain ratio and presented to the user. This process is repeated after each interaction between the user and the search engine.

Even though the previously discussed approaches are specifically geared towards e-commerce, they have various shortcomings. Approaches such as [17], which rely on well-estimated user statistics, are not usable in practice. The reason for this is that some Web shops do not have the infrastructure to collect the user data or that the product catalogs are relatively large and changing rapidly. Other methods, such as [15, 16], do not rely on previously collected user statistics, but they suffer from other issues. One of the main arguments against these example critiquing approaches is that they deviate too much from the currently used interaction paradigms (i.e., regular faceted navigation). We argue that an approach that lies closer to what users use on a daily basis will be more successful when deployed in practice. Therefore, our focus is to have an approach that introduces minimal changes to the classical faceted navigation interface present in current Web shops, while supporting approximate search and explicit user preferences. The only user interface adaption we make is that users can specify importance weights for the selected facets, which can be achieved in many different, non-intrusive ways.

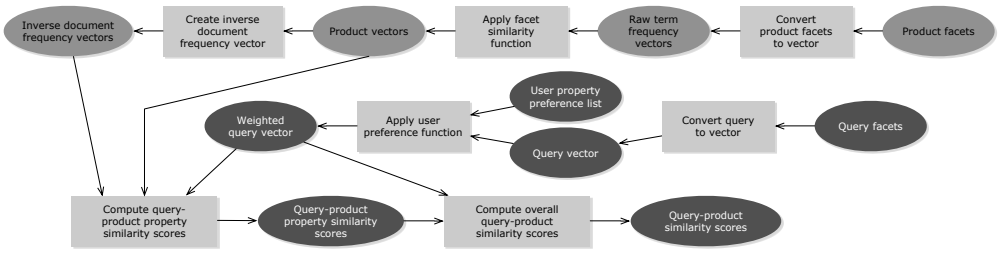

Figure 1: Overview of the framework.

3. Approximate Product Search

Before diving into the details of the framework, we first discuss facets in more detail and give an overview of all the framework components. Product information in Web shops is usually stored

in the form of tabular data, consisting of key-value pairs, where the key denotes the *property* to which the value belongs. Each distinct combination of a key and value occurring in the product catalog is considered to be a facet. For example, a key-value pair, or facet, of a product might be `['Colour', 'Black']`. Later on, if there is no ambiguity, we use the term 'value' to refer to a facet, in order to shorten the explanations. There are various property types, which need to be treated differently, both in the front-end as well as in the back-end. There are two main types of properties, namely qualitative and quantitative properties.

Qualitative properties are further discerned based on whether they allow one or multiple values per product. Typically these two types are presented differently to the user, either in the form of radio buttons, which only allow one selection, or a list of check boxes that allows multiple selections. An example of the first type of qualitative property is a qualitative Boolean property, e.g., the key-value pair `['WiFi', 'True']`. In contrast, a key-value pair like `['Supported Audio Formats', 'MP3, FLAC, AAC']` allows multiple values for a product. In this case, each element in the list of values should be treated as a separate facet. There is an important exception to how single-valued qualitative properties other than Boolean qualitative properties are treated. These are namely still treated as a multi-valued property in user interfaces. One example of this would be the 'Brand' property. The reason for this is that users may wish to include products from multiple brands in the query (e.g., in order to compare them). Therefore, we treat multiple selections of facets from a property like 'Brand' as a generalized disjunctive query. This is consistent with the interpretation that many major Web shops employ, such as Amazon.com and BestBuy.com. Quantitative properties have a single numerical value, where every distinct numerical value is considered to be a facet. Different from boolean properties, a user can select multiple values for a quantitative property by selecting a range of values. For example, the user could search for a product with a price between €100 and €200, which encompasses multiple facets. Then, all the products which have a price within this range will receive a weight of 1 for that facet (meaning the facet is present).

Queries submitted to a faceted search engine consist of a selection of facets, from which a Boolean expression is constructed. Traditionally the 'Standard Boolean Model' is used to evaluate the expression, which means that a product is only included in the result set if the Boolean expression evaluates to true for the product. Our proposed algorithm replaces this strict model with an

'Extended Boolean Model' that computes a similarity score for each product, thus enabling approximate search. The constructed query expression is usually made disjunctive for facets belonging to the same property, and conjunctive for facets belonging to different properties. Large Web shops, such as Amazon.com, employ this principle, as it is considered to be intuitive to the users to form the query in this way. For example, choosing `['WiFi', 'True']` and `['Camera Flash Type', 'Single LED, Dual LED, Xenon']` would search for all products that have WiFi and a Camera Flash Type of either 'Single LED', 'Dual LED', or 'Xenon'.

3.1. Framework Overview

The proposed algorithm employs a similarity score between the query and each product in the catalog, which indicates how closely a product matches the query. In order to be able to compute this score, it is necessary to convert both products and query to vectors first. Note that each distinct numerical value of a quantitative property is treated as a separate facet, and is therefore assigned to an element in the vector representation.

Figure 1 illustrates all the framework processes involved to compute the product-query similarities. In this figure, rectangles represent processes and ovals represent input/output of a processes. Furthermore, darker-gray ovals are dependent on the query, while light-gray ovals are not. This means that the light-gray processes can be computed in advance to improve the computational time.

First, let us discuss the process of converting *product facets* to *product vectors* and *inverse document frequency vectors* (i.e., the first row of the figure). The facets associated with a product are first converted to a binary weight vector, which is called the *raw term frequency vector*. A weight of zero at index i indicates that the product does not contain the corresponding facet, whereas a weight of one means that it does. Afterwards, a *facet similarity function* is applied to each element in the raw term frequency vector that has a value of zero. This function converts the binary weights to weights that range from zero to one. This can be interpreted as to how similar the product is to the considered missing facet (based on all available product data). A different weighting approach is used for the various facets in the vector, depending on whether the facet corresponds to a qualitative or quantitative property. We will discuss this process in more detail in Section 3.2. The last step in this part of the flow is to convert the product vector to the classical inverse document frequency vectors [35].

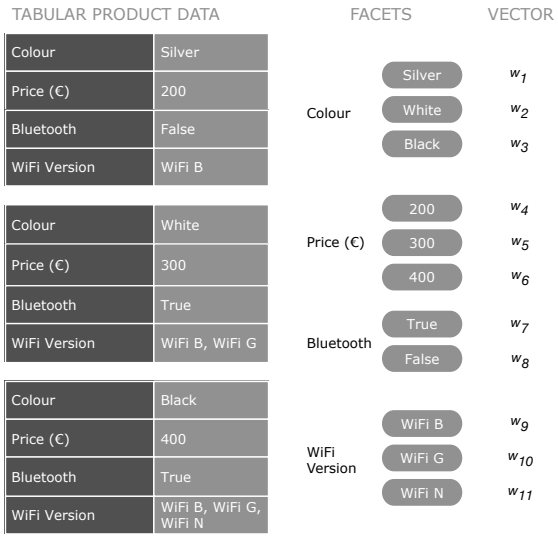

Figure 2: Three example products and their facets.

On the second row of Figure 1 we see the processes that encompass the query flow, converting *query facets* to a *weighted query vector*. All of these processes are dependent on the query and have to be computed each time a query is submitted to the search engine. Every time a query is submitted to the search engine, the algorithm first converts it to a vector, which has the same dimensions as the previously created vectors for products. Similar to the raw term frequency vectors, a weight of zero at index i indicates that the query does not contain the corresponding facet, whereas a weight of one means that it does. A quantitative facet is contained in the query if its associated numerical value is within the query range for the associated property.

In addition to the query facets, the user also submits a *user property preference list* to the search engine. This preference list consists of all the involved properties in the facet selections and is ranked descending on the importance (determined by the user). The *user preference function* takes the query vector and the user preference list as input to determine the *weighted query vector*. The weight query vector is computed by multiplying each value in the query vector with a property importance weight. The details of this procedure are discussed in Section 3.3.

After both the products and the query have been converted to a vector, it is possible to compute the similarity between them. This is depicted on the third row of Figure 1. The similarity score is calculated using the *p-norm extended Boolean model* [12], and is a two-staged process. First, the similarity of each property in the query is computed for every product using the p-

norm extended Boolean formula for disjunctive queries. Second, the overall similarity between the query and each product is computed. This step uses the similarity scores for each property from the previous step and weights them again according to their associated weights in the weighted query vector. Afterwards, these weights are plugged into the p-norm extended Boolean formula for conjunctive queries, which results in an overall similarity score. These steps are described in detail in Section 3.4. The end result of the process is a similarity score between zero and one for each product, which indicates how closely a product matches the query. Last, the products are sorted on their similarity scores in descending order and are presented to the user.

We move on now with the discussion of each of the processes depicted in Figure 1 in more detail. Table 1 provides an overview of the notations that are used throughout the rest of this paper.

3.2. Product IDF Vectors

We start with the detailed explanation of the process of converting *Product Facets* to *Product Vectors* and *Inverse document frequency vectors*, i.e., the first row of Figure 1. The first step in converting product data to vectors consists of extracting the facets from the available tabular product data. Figure 2 depicts an example of the facet extraction process with three products. Each distinct key-value pair is converted to a facet and is assigned a vector index. The vector index of every facet corresponds to an element within the vector. Using the extracted facets for every product, we construct *raw term frequency vectors*. These vectors are straightforward binary vectors indicating if a facet is present or not. For example, for product 1 in Figure 2 the raw term frequency vector would be $(1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0)$.

After the raw term frequency vectors have been created, the *facet similarity function* is applied to each weight in the vector. This conversion is needed as binary weights are too strict for the purpose of approximate product search.

Ideally we want the search engine to rank and not filter the products using the submitted query. For example, consider a user who is searching for a product with a price between €100 and €200. Now suppose that a product exists with a price of €99.95. As the price of the product does not fall within the specified range, the product does not match the query and thus gets a low similarity score or is even not included in the results at all. Intuitively this could be considered as an incorrect decision, as the price nearly matches the lower bound of the range, thus the product might be

interesting to the user as well. Besides being too strict, using binary weights also gives another problem: any quantitative value that lie within the specified range is considered equally dissimilar from the query. This means that, for a query where the target price is €100, a product with a price of €500 would achieve the same similarity score as the product with a price of €99.95 (all other aspects being equal), which is clearly undesirable, as the latter product should be considered as being more similar to the query.

The same problems also arise when searching for products with a qualitative property. If a product does not contain the same qualitative facets as the query, it gets excluded. Furthermore, qualitative facets that are not included in the query are treated as equally dissimilar from those in the query. This does not always make sense, as some facets are conceptually quite similar, whereas others are not. For example, consider some of the audio file formats that exist, such as 'MP3', 'Ogg Vorbis', and 'WAV'. Whereas 'WAV' is an uncompressed and lossless format, both 'MP3' and 'Ogg Vorbis' are examples of a compressed and lossy format. Thus it can be argued that the latter two audio formats are more similar to each other than 'WAV' and either one of these formats.

The facet similarity function aims to address the issues outlined above by converting the weights in the raw term frequency vector to a weight in the range of zero to one, which can be interpreted as the degree of similarity between a product and each facet. The result of applying the function on the raw term frequency vector is called the *product vector*. Depending on whether a facet belongs to a quantitative or quantitative property, a different approach is used to compute its weight in the product vector. This similarity function is defined as:

$$\text{facetSim}(f_p, d) = \begin{cases} 0 & \text{if } F_{p,d} = \emptyset \\ 1 - \min\limits_{f_{p,d} \in F_{p,d}} \dfrac{|f_p - f_{p,d}|}{\max(F_p) - \min(F_p)} & \text{if } p \in P_{\text{quan.}} \\ \max\limits_{f_{p,d} \in F_{p,d}} \dfrac{|D_{f_p} \cap D_{f_{p,d}}|}{|D_{f_{p,d}}|} & \text{if } p \in P_{\text{qual.}} \end{cases} \tag{1}$$

Evidently, if a product does not have any of facets associated with the property p, the similarity score remains zero, as it is not possible to compute the degree of similarity in that case.

The score for quantitative properties is based on the linear distance between the numerical value of the facet f_p and the numerical value of the product. Note that, although $F_{p,d}$ is defined as a set, there is often only one element in the set for quantitative properties, as a product usually does not have multiple numerical values for the same property. The distance between the numerical

values is normalized by dividing it by the range of the values for the property, thus obtaining a relative dissimilarity. By subtracting the dissimilarity from one, the similarity score between two quantitative facets is obtained.

The facet similarity score for qualitative properties computes a similarity score between the facet f_p and each facet of property p that is also associated with product d, i.e., each facet $f_{p,d}$ in $F_{p,d}$. The similarity score is based on the fraction of products with facet $f_{p,d}$ that are also associated with facet f_p. After computing all the similarity scores, the highest score is selected and is used as the weight for facet f_p in the product vector.

The rationale behind this score is that facets that occur frequently together within product data are likely to be conceptually similar. For instance, consider the different versions of the WiFi standard, such as 'WiFi B', 'WiFi G', and 'WiFi N'. Each consecutive version of the standard enhances the previous version and is backwards compatible, thus products adhering to a particular version also adhere to the previous versions. Suppose that the user is searching for a product with 'WiFi N'. Then products which do not have 'WiFi N', but do have 'WiFi G', are a better alternative than products that only have 'WiFi B', as the versions are conceptually more similar. For example, suppose that there are 20, 10, and 5 products associated with 'WiFi B', 'WiFi G', and 'WiFi N', respectively, and each product with 'WiFi N' also has 'WiFi B' and 'WiFi G'. Then the similarity score for the facet 'WiFi N' would be $\frac{5}{20}$ $(= \frac{1}{4})$ for products with only 'WiFi B', and $\frac{5}{10} (= \frac{1}{2})$ for products with 'WiFi G'.

By applying (1) to each weight in the raw term frequency vector, a new vector, called the product vector, is created with weights ranging from zero to one. Using the example products from Figure 2 again, we obtain the following product vectors:

$$productVector_1 = \left(1, 0, 0, 1, \frac{1}{2}, 0, 0, 1, 1, \frac{2}{3}, \frac{1}{3}\right)$$
$$productVector_2 = \left(0, 1, 0, \frac{1}{2}, 1, \frac{1}{2}, 1, 0, 1, 1, \frac{1}{2}\right)$$
$$productVector_3 = \left(0, 0, 1, 0, \frac{1}{2}, 1, 1, 0, 1, 1, 1\right)$$

After the product vectors have been created, the *inverse document frequency vectors* are computed [36]. Inverse document frequency vectors are useful for faceted search, as it provides the means to weight facets in the query, based on how often they occur within the product catalog.

D	Product catalog
P	Properties
$P_{\text{qual.}} \subseteq P$	Qualitative properties
$P_{\text{quant.}} \subseteq P$	Quantitative properties
$F_p \subseteq F,\ p \in P$	Facets for property p
$f_p \in F_p$	Facet for property p
$F_d \subseteq F,\ d \in D$	Facets for product d
$f_d \in F_d$	Facet for product d
$F_{p,d} = F_p \cap F_d$	Property p facets for product d
$f_{p,d} \in F_{p,d}$	Property p facet for product d
$D_f,\ \forall d \in D_f : f \in F_d$	Products d associated with facet f
$q \subseteq F$	Query
$D_q \subseteq D$	Result set returned for query q

Table 1: Summary of used notations.

Products that have the more uncommon facets in the query will therefore receive a higher score than products which do not have those facets. For example, if a user searches for a product that has Bluetooth and is dust resistant, it is more likely that products with dust resistance are more important to the user than products that only have Bluetooth, as this facet occurs less frequently in the catalog.

Even though the idea of using the inverse document frequency can be useful in our context, we propose some adaptations to make it fit better. We define the inverse document frequency of item i (a facet) as following:

$$\text{idf}(i) = \frac{\log\left(N / \sum_{n=1}^{N} w_{n,i}\right)}{\log N} \tag{2}$$

where $w_{n,i}$ corresponds to the value for item i for product n and N is the number of products in the catalog.

First, the p-norm extended Boolean model that is used by the framework requires weights to be in the range of zero to one. However, this is currently not the case, as weights can be any positive number. Therefore, the weights are normalized through dividing by $\log N$.

In addition, the term n in the original idf formula is exchanged for a slightly different term.

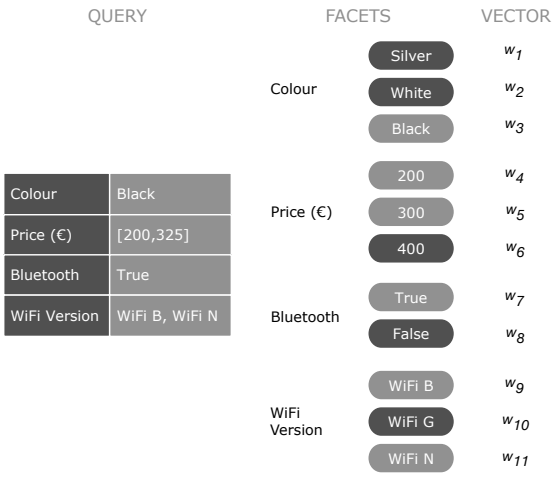

Figure 3: A query and the corresponding facets.

Rather than being equal to the number of products that are associated with a facet, the new term is defined as the sum of facet similarities of all products. In other words, it is a summation of the facet weights in all product vectors. This is done in order to prevent a bias towards facets associated with few products, i.e., the size of the set D_f is quite small, while many products do have a high facet similarity score for such a facet. In this way, we obtain inverse document frequency weights for these facets that better reflect how often they occur within the catalog, as facet similarities are now taken into account. The inverse document frequency vector for the first example product vector from Figure 2 is:

$$\left(\frac{\log 3/1}{\log 3}, \frac{\log 3/1}{\log 3}, \frac{\log 3/1}{\log 3}, \frac{\log 3/1\frac{1}{2}}{\log 3}, \frac{\log 3/2}{\log 3}, \frac{\log 3/1\frac{1}{2}}{\log 3}, \frac{\log 3/2}{\log 3}, \frac{\log 3/1}{\log 3}, \frac{\log 3/3}{\log 3}, \frac{\log 3/2\frac{2}{3}}{\log 3}, \frac{\log 3/1\frac{5}{6}}{\log 3}\right) \quad (3)$$

As it becomes clear from the previous example, a facet weight is equal to zero when all the products in the catalog have a score of one for the facet in their product vector, whereas a facet weight is equal to one when only a single product has a non-zero score for the facet in the product vector. This is logical, because including a facet in the query that every product has does not provide any extra information to the search engine. Conversely, when only one product is associated with a facet, it provides the maximum amount of information gain to the search engine.

3.3. Weighted Query Vectors

The process of converting a query to a vector is very similar to the process used to convert a product to a vector. Similar to product data, we extract facets from a submitted query. There is a small difference in how quantitative key-value pairs are handled though. Rather than having a

single numerical value, queries have a numerical range as value for a quantitative property. Figure 3 illustrates how this process works with an example. The dark-gray facets are not included in the query, whereas the light-gray facets are included. For example, the price range $[200, 325]$ is mapped to the facets '200' and '300', but not to '400', as that value is out of range. The obtained query vector consists of binary weights, where a weight of zero means that the corresponding facet is not included in the query, and a weight of one means that it is included. For example, the query vector for the query in Figure 3 is defined as $(0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1)$.

With the (binary) query vectors, every facet is regarded as being equally important to the user. However, it can be argued that this is often not the case, as the user has both hard and soft requirements for a desired product. Therefore, the weights in the query vector should be adjusted according to the user preferences, in such a way that it becomes possible to make a distinction between hard and soft requirements.

Assigning correct weights to the facets can partially be accomplished by using the inverse document frequency vector, as previously discussed. However, as these weights are solely based on the product data available in the catalog, the result is a vector that is the same for every user. It is therefore unsuitable for taking the user preferences into account, as not every user who submits a query has the same requirements. That is why instead we propose to extend the query by including a ranked list of properties, the *user property preference list*, which ranks the properties in descending order of importance to the user.

Every property p that occurs within the query, i.e., there is at least one facet f_p in query q, is included in the list. The reason for aggregating the preferences on a property level rather than on an individual facet level is twofold. First, queries submitted to a faceted search engine are typically quite large in size and therefore rather complex. Having to manage a list of preferences should not be too obtrusive for the user during the querying process. This is why the list should ideally contain only a few elements, even when the query contains many facets. Aggregating on a property level ensures that the preference list stays concise and manageable while the user gradually refines his query. Second, the query itself is also structured in a similar way, as it is disjunctive for facets belonging to the same property, and conjunctive between properties. This suggests that facets belonging to the same property are of equal relative importance to the user, whereas facets belonging to different properties are not. This usually reflects the user intentions better than a

fully conjunctive or disjunctive query. It therefore makes sense to structure the user preferences in the same way.

The equation for converting a facet weight from the query vector to a weight that takes user preferences into account is given by:

$$\text{weight}(w_i, p) = \frac{1}{\text{rank}(p)} \cdot w_i \tag{4}$$

where w_i is the facet weight in the query vector, p is the property to which the facet belongs, and `rank(`p`)` is a function that returns the rank of property p in the user property preference list. The old weight w_i in the query vector is multiplied with the simplest form of Zipf's law, which was first proposed in [37], in order to obtain a new weight that takes the user preferences into account. Suppose that the user preference list of the user, who submitted the example query in Figure 3, is equal to `['Colour', 'Bluetooth', 'Price', 'WiFi Version']`, then applying the user preference function to the query vector yields the weighted query vector $\left(0, 0, 1, \frac{1}{3}, \frac{1}{3}, 0, \frac{1}{2}, 0, \frac{1}{4}, 0, \frac{1}{4}\right)$.

3.4. Query-Product Similarity Score

Now that both the products and the query are converted to vectors, we compute a similarity score between each product and the query. The similarity score is calculated using the p-norm extended Boolean model. The framework first computes the similarity scores for each property, followed by the overall similarity score.

The standard Boolean model is the usual method for querying faceted product data. Each submitted query is converted to a Boolean expression and only products that completely match the expression are returned. This method has several drawbacks that are mostly related to the strictness of the model. Because products are only returned if they fully match the query, it is highly error-prone and unsuitable for exploratory search, as the user has to constantly correct the query when no products are returned. As is evident from the example product data in Figure 2, none of the products match the query. However, there are some products that nearly match the query, but have the wrong color or their price is too high. Ideally, these trade-off possibilities should be presented to the user without the user having to make his query less specific first.

In addition, the strictness of the model also does not allow for user preferences to be taken into account. The user might have both soft and hard requirements for desired facets, but the standard Boolean model only returns products that completely match the query. It is therefore

not possible to indicate that a particular term in the query is more important than another, as they are inherently all equally important.

Another deficiency of the standard Boolean model is the implicit lack of ranking within the result set. As each product in the result set matches the query, it is not possible to distinguish between them. This means that a product, which has both 'WiFi B' and 'WiFi N', is not ranked higher than a product with only 'WiFi B' or 'WiFi N' for the example query in Figure 3, while, using generalized disjunction, it could be argued that it would preferable to do so.

It is clear that the standard Boolean model is not suitable for approximate product search, which means that an alternative model has to be used. One of the alternatives is the vector space model [38]. Although this model does allow term weighting, the query is interpreted as a fully conjunctive query. This is obviously not ideal for faceted product search, as usually queries consist of conjunctive and disjunctive terms.

Therefore, the p-norm extended Boolean model [12] is used by the framework, which offers both the advantages of the vector space model and supports conjunctive queries. It essentially softens the strict *AND* and *OR* Boolean logic operators of the standard Boolean model by calculating a distance between the desired values, i.e., the values in the query, and the values within a product vector. This distance is based on the p-norm distance, a generalization of the Euclidean and Manhattan distance measures, which includes a parameter p to vary the degree of strictness of the operator. In our approach, we use a value of two for the p parameter, which is a well-known and frequently used norm called the Euclidean norm. The results in [12] indicate that a value between two and five is optimal, whereas larger values decrease the performance improvements over the standard Boolean model. For disjunctive queries, the similarity for an n-dimensional vector using the p-norm extended Boolean model is given by:

$$\text{sim}(\mathbf{q}, \mathbf{d}) = \sqrt[p]{\frac{\mathbf{q}_1^p \cdot \mathbf{d}_1^p + \mathbf{q}_2^p \cdot \mathbf{d}_2^p + \cdots + \mathbf{q}_n^p \cdot \mathbf{d}_n^p}{\mathbf{q}_1^p + \mathbf{q}_2^p + \cdots + \mathbf{q}_n^p}} \tag{5}$$

where p is the p-norm parameter, q is the query vector, and d is the document vector. The similarity for a conjunctive query and an n-dimensional vector is given by:

$$\text{sim}(\mathbf{q}, \mathbf{d}) = 1 - \sqrt[p]{\frac{\mathbf{q}_1^p \cdot (1 - \mathbf{d}_1)^p + \mathbf{q}_2^p \cdot (1 - \mathbf{d}_2)^p + \cdots + \mathbf{q}_n^p \cdot (1 - \mathbf{d}_n)^p}{\mathbf{q}_1^p + \mathbf{q}_2^p + \cdots + \mathbf{q}_n^p}} \tag{6}$$

where p is the p-norm parameter, q is the query vector, and d is the document vector.

The first step in calculating the query-product similarity scores is calculating the similarity score per property. As it is assumed that queries between facets from the same property are disjunctive, Equation (5) is used for this step. Because the score is calculated for each property p, a sub-vector $\mathbf{d}$ is created from the product vector including only facets associated with property p. The same process is applied to the weighted query vector and the inverse document frequency vector, resulting in sub-vectors $\mathbf{q}$ and $\mathbf{idf}$, respectively. Last, the similarity score between $\mathbf{d}$ and $\mathbf{q}$ is computed, resulting in a score between zero and one.

Note that the weights for each facet in the query vector $\mathbf{q}$ are all equal, as they belong to the same property and the preference weighting is performed on a property-level. However, it can be argued that the facets should be weighted, as some facets are more unique than others, thus a product having that facet should be regarded as being more similar to the query. The reason for this distinction is that the user included an uncommon facet in the query, which could indicate that it is an important facet to him. For example, suppose that a query contains the facets 'WiFi G' and 'WiFi N' associated with the property 'WiFi Version'. As the versions are backwards compatible, there are relatively many products with 'WiFi G' compared to products with 'WiFi N'. In this situation it is more likely that the user would prefer products which have 'WiFi N' rather than products with only 'WiFi G'. Therefore, the query vector weights are multiplied by the inverse document frequency weights, in order to assign different weights to the facets in the query. The equation is therefore given by:

$$\text{propSim}(\mathbf{q^i}, \mathbf{d^i}) = \sqrt[p]{\frac{(idf_1^i \cdot q_1^i)^p \cdot {d_1^i}^p + \cdots + (idf_n^i \cdot {q^i}_n)^p \cdot {d_n^i}^p}{(idf_1^i \cdot q_1^i)^p + \cdots + (idf_n^i \cdot q_n^i)^p}} \tag{7}$$

where p is the p-norm parameter and q_j^i, idf_j^i, and d_j^i are the query, the inverse document frequency, and the product vector values for property i and vector index j (the vector corresponding only to property i), respectively.

Note that the query weights in the denominator are also multiplied by the inverse document frequency. This is done in order to keep the score normalized to a maximum value of one. Otherwise, even when the document would have all the facets in the query, i.e., $\mathbf{d}$ consists only of values of one, the resulting score might not be equal to one, which is counter-intuitive. In this case, the numerator can only be equal to the denominator, thus resulting in a score of one, if the inverse document frequency vector also consists only of values of one, which would mean that all the facets may only occur once in the product catalog. We therefore normalize the score by

also multiplying the query vector weights in the denominator with the inverse document frequency weights. Consider the example weighted query vector

$$\left(0, 0, 1, \frac{1}{3}, \frac{1}{3}, 0, \frac{1}{2}, 0, \frac{1}{4}, 0, \frac{1}{4}\right),$$

and the first example product vector

$$\left(1, 0, 0, 1, \frac{1}{2}, 0, 0, 1, 1, \frac{2}{3}, \frac{1}{3}\right),$$

and the inverse document frequency vector in Equation (3). After decomposing each vector into the sub-vectors for each property, as previously discussed, the property similarity scores are computed as follows:

$$\begin{aligned}
\text{propSim}(\mathbf{q_p}, \mathbf{d_p}) &\approx 0.90 \\
\text{propSim}(\mathbf{q_w}, \mathbf{d_w}) &\approx 0.33 \\
\text{propSim}(\mathbf{q_c}, \mathbf{d_c}) = \text{propSim}(\mathbf{q_b}, \mathbf{d_b}) &= 0
\end{aligned}$$

where c, p, b, and w are the properties 'Colour', 'Price (€)', 'Bluetooth', and 'WiFi Version', respectively.

Looking at the property similarity scores, we can observe that the product has nothing in common with the query regarding the properties 'Colour' and 'Bluetooth'. This is due to the fact that the product only has non-zero scores in its product vector for facets that are not included in the query. In other words, it has a wrong value for 'Colour', namely 'Silver' instead of 'Black', and it does not have Bluetooth. Furthermore, because there are no other products associated with 'Silver' or 'False' that are also associated with 'Black' or 'True', the weights of the product for 'Black' and 'True' remain zero in the product vector, as there is no facet similarity. In contrast, the product achieves a high score for the property 'Price (€)', as its value (200) is within the range of the query ([200,325]).

While the product does have one out of two of the facets associated with the property 'WiFi Version' in the query, it only gets a score of 0.33. This is due to the fact that it only has a score of one in its product vector for the most common 'WiFi Version', namely 'WiFi B'. As all products have a score of one for that facet, its weight in the inverse document frequency vector is zero, as it conveys no information regarding the product that the user wants to buy. However, because some of the products that have 'WiFi B' also have 'WiFi N' associated with them, the product has a

weight of 1/3 associated with 'WiFi N' in its product vector. Therefore, the property similarity score of this product is not equal to zero, but equal to 1/3.

After the property similarity scores have been computed, the *overall similarity score* can be computed. As the query is assumed to be conjunctive between properties, the framework uses Equation (6) for this step. Furthermore, it uses the property similarity scores for each property as the weights for the product.

Because the facets are now aggregated on a property level, it is also necessary to aggregate the weights in the vectors. First, the property similarity scores are combined to form the aggregated product vector. Using the decomposed property similarities for the first product vector in Figure 2, we determine this vector to be $(0, 0.8993, 0, 0.33)$. Then, the similarities for each property have to be weighted, such that products with a high similarity score for important properties are promoted in the search results. As previously discussed, a list of properties in the query, ranked in decreasing order of importance, is obtained from the user. Using this list, Equation (4) is applied to each weight in the query vector in order to transform it to the weighted query vector. Subsequently, the highest weight per property in the weighted query vector is collected and together they form the weighted aggregated query vector. For the example weighted query vector, this vector is $\left(1, \frac{1}{3}, \frac{1}{2}, \frac{1}{4}\right)$. After the aggregated query and product vectors have been obtained, the overall similarity between them can be computed as follows:

$$\text{overallSim}(\mathbf{q}, \mathbf{d}) = \sqrt[p]{\frac{\mathbf{q}_1^p \cdot \mathbf{d}_1^p + \mathbf{q}_2^p \cdot \mathbf{d}_2^p + \cdots + \mathbf{q}_n^p \cdot \mathbf{d}_n^p}{\mathbf{q}_1^p + \mathbf{q}_2^p + \cdots + \mathbf{q}_n^p}} \tag{8}$$

where p is the p-norm parameter and q is the query vector and d is the document vector. For the first example product vector $(0, 0.8993, 0, 0.33)$ and the weighted example query vector $\left(1, \frac{1}{3}, \frac{1}{2}, \frac{1}{4}\right)$ the overall similarity score is computed as following:

$$\text{overallSim}(\mathbf{q}, \mathbf{d}) = \sqrt{\frac{1^2 \cdot 0^2 + \frac{1}{3}^2 \cdot 0.8993^2 + 0^2 \cdot \frac{1}{2}^2 + \frac{1}{4}^2 \cdot 0.33^2}{1^2 + \frac{1}{3}^2 + \frac{1}{2}^2 + \frac{1}{4}^2}} \approx 0.0522$$

As we can see, the high similarity score for one of the properties is offset by low scores for the more important properties, thus the product receives a low overall similarity score of 0.05.

The final scores for the the three products from Figure 2 is 0.05, 0.15, and 0.79, respectively. Note that, as none of the products has achieved a perfect overall similarity score, no products would have been returned if the standard Boolean model was used. This would result in the user having to adjust his query in order to find trade-off possibilities, whereas this is not needed in the proposed framework. Clearly the third product matches the query better than the other products,

due to having perfect similarity scores for three of the four properties in the query. It therefore matches most of the user's requirements, apart from the price, which is too high. However, the user has indicated previously that the price is a relatively soft requirement for him. The user preference function has taken this into account by assigning a weight of 1/3 to the price property, which helps boost the overall similarity score of the third product. This assists the user in finding the best trade-off possibilities, given his preferences, if there is no product that completely matches his query.

4. Evaluation

For evaluating the performance of the algorithms, we use a real-life dataset from [13], which is a commercial Web site with a large community consisting of more than half a million Dutch and Belgian technology enthusiasts, collected in 2014 for our previous work [21]. We have chosen to focus the research on faceted search within a product catalog consisting entirely of mobile phones. The reason for this is that products in this category differ greatly in functionality and appearance, introducing heterogeneity in the data. The data of 794 different mobile phones was obtained from the Pricewatch using a combination of tools and scripts.

4.1. Experimental Setup Using Simulations

For our experiments we use the interaction model proposed in [39], where the user is presented a list of facets and iteratively updates the query by selecting one or more facets. In our setup, a search session is defined as an interactive process of updating the query, until the user has no desire to further refine the query. More specifically, first, the user decides whether the query should be updated any further, which is based on the number of user actions executed so far, and the total number of user actions that he is willing to perform. If the user wants to update the query, the user considers the generated property preference list first, a ranked list of properties in the query in descending order of importance. The user compares the generated user preference list with the actual property preference list, which reflects his or her preferences regarding the properties associated with the target product. If the user spots an inconsistency within the ranking of the properties in actual and generated user preference list, the generated list is updated by dragging the first incorrectly ranked property to the desired position. Afterwards, the preference list is

submitted to the search engine, which subsequently updates the result set by taking the new user preferences into account.

If the user cannot perform preference reordering, either because the ranking between the actual and generated user preference list is already consistent, or the employed algorithm does not take user preferences into account, the query is updated by selecting an additional facet instead. It is assumed that the user scans the list of displayed facets in a linear order, from the top to the bottom. The ordering of the list is based on the expert-based ordering employed by [13], and does not change during the search session. Every time a facet is examined, the user first tries to identify it as a facet belonging to the target product. Rather than being able to identify each facet of the target product with full accuracy, the user has only a chance of α to correctly identify it. Conversely, the user has a chance of β to falsely identify a facet as belonging to the target product. For our evaluation, we use $\alpha = 0.9$ and $\beta = 0.1$.

In order to prevent a bias towards examining facets from properties which have many facets, we distribute the α and β chances over all the facets per property. Otherwise, a facet from a property with many facets, such as 'Colour', would be included more often in the query than a facet from a Boolean qualitative property, such as 'Bluetooth'. Therefore, the probability to consider selecting a facet is α_f or β_f, depending on whether the facet is associated with the product.

After identifying a facet as belonging to the target product, either correctly or incorrectly, the user will consider whether he wants to actually select the facet for inclusion in the query. After the user has made a selection, the updated query and generated preference list are submitted to the search engine and the number of user actions is incremented. The search engine subsequently computes new similarity scores between all the products and the query, and returns a new ranked result set. This process continues until the user is no longer willing to perform more user actions, at which point the simulated search session is concluded.

We vary the total number of clicks per search session using 5, 10, 20, and 30 clicks. Furthermore, the target product, which the user is looking for, is also varied. The combination of algorithm, target product, and number of clicks are used as inputs to the evaluation framework. Each combination of inputs for simulating a search session is defined as one experiment. Note that each experiment is executed fifty times, in order to reduce the stochasticity, which is inherent in the drill-down model that simulates a user.

In addition to our proposed algorithm, the performance of several other algorithms is measured and compared to each other. The first algorithm, which is used as a baseline, employs a simple similarity function for each product, based on the number of facets in the query that the product matches. The p-norm extended Boolean model [12] forms the basis of our proposed algorithm. It therefore makes sense to compare the results of the proposed algorithm with those of the regular p-norm extended Boolean model. A variation of the proposed algorithm that does not take user preferences into account is the third algorithm against which we compare the performance of our approach.

Last, for our experiments we needed to model the user behavior when interacting with the faceted search engine. For this purpose, we base our assumptions on the ones previously outlined in [25, 40, 41]: rationality, omnisciency, linearity, and practicality. For more details on these assumptions we refer the reader to [25, 40, 41]. In our approach we relax the omnisciency and linearity assumptions by introducing some stochasticity to the model when selecting facets to include in the query, which is implemented differently for qualitative and quantitative facets.

4.2. Results Using Simulations

Various performance measures are collected during the experiments. In order to assess the performance of the algorithms, it is useful to gather some measures related to the position of the target product. First, we define the 'last position' as the last position of the target product after T user actions. This function returns the rank of the target product d_u in the result set D_q. Second, we define the 'average position' as the average position of the target product computed after each user action. Third, we use the 'first iteration in top-N' metric to measure how quickly an algorithm manages to promote the target product to the first top N results. Similarly, the 'any iteration in top-N percentage' metric measures the percentage of sessions in which the target product was promoted at least once to the top N results. Last, the 'success percentage' indicates in how many search sessions the target product was in the top-N results after T user actions. In addition to collecting performance measures related to the position of the target product in the result set, we also collect measures regarding the user effort that is required to formulate the query. The user effort involved can be split up into various components, related to the various tasks a user performs while formulating the query. The first task we define is the reordering of the user property preference list. The other two tasks are related to scanning the list of displayed

properties P (property scan effort) and the list of displayed facets F (value scan effort). Last, the computation time of the algorithm required for computing the ranking of the products in the result set is also measured. The time is measured in milliseconds and consists only of the time to compute the ranking. The time needed to simulate the user's behavior is therefore not included in this measure.

As previously outlined, all combinations of an algorithm, a target product, and a total number of clicks were executed, with each experiment being conducted fifty times. This means that a total of 635,200 search sessions were simulated (4 algorithms $\times$ 794 products $\times$ 4 total clicks $\times$ 50 repetitions). Because each search session consists of 16.25 ($= \frac{5+10+20+30}{4}$) user actions on average, it results in a total of 10,322,000 simulated user actions. Due to these vast numbers, the experiments were run in parallel on a cluster of server nodes [42], in order to speed up the evaluation.

Table 2 shows an overview of the obtained performance measures for the four algorithms and for $T = \{5, 10, 20\}$. We do not discuss the case of $T = 30$ for the sake brevity, however, we can report that the relative results are the same as for $T = 20$. Only the absolute differences between the algorithms became larger. At first sight, from Table 2 we can see that for $T = 5$, the algorithms perform fairly similar. This is because the queries are relatively homogeneous when only five clicks are used, making the effect of the ranking algorithms less clearly visible. However, we do make some interesting observations. First, we notice that the results for the 'simple rank' algorithm are relatively good. It outperforms all other algorithms for the last position metric ($p < 0.00001$). With respect to the success percentage metric, it outperforms our approach with user preference ranking but performs worse than our approach without user preference ranking (all differences are significant with $p < 0.00001$). After analyzing the reason for this behavior we discovered that the main disadvantage of the 'simple rank' algorithm, i.e., not distinguishing between conjunctive and disjunctive aspect in a query, is not visible when limiting the number of clicks to 5. The reason for this is that the facets are scanned linearly and the first five facets belong to properties for which almost all products only have one value. These facets are 'Price', 'Brand', 'Operating System (OS)', 'OS version', and 'Color'. As a result, in almost all simulated sessions, a conjunctive query was performed, therefore not allowing algorithms that support disjunctive queries to distinguish themselves.

Second, we notice that the p-norm approach has a low success percentage, with a significant

	Simple Rank	P-norm	Our Algorithm without Preferences	Our Algorithm with Preferences
For $T = 5$:				
Last Position Mean	84.25	95.05	103.66	102.14
Avg. Position Mean	152.94	159.58	141.77	145.95
Any Iteration Top-N Percentage	44.74%	44.53%	55.47%	44.03%
First Iteration Top-N Mean	3.34	3.29	3.01	2.89
Success Percentage	40.23%	37.01%	44.86%	36.56%
Preference Reorder Count Mean	–	–	–	1.1
Property Scan Effort Mean	0.0986	0.0988	0.0987	0.0706
Value Scan Effort Mean	0.2652	0.2635	0.2625	0.1844
Computation Time Mean (ms)	1380	2232	1207	1199
For $T = 10$:				
Last Position Mean	75.11	93.01	122.84	79.25
Avg. Position Mean	113.10	124.44	126.75	114.05
Any Iteration Top-N Percentage	68.78%	68.99%	71.96%	66.87%
First Iteration Top-N Mean	4.76	4.71	3.97	4.47
Success Percentage	57.73%	53.30%	51.85%	53.78%
Preference Reorder Count Mean	–	–	–	2.8
Property Scan Effort Mean	0.1287	0.1288	0.1286	0.0822
Value Scan Effort Mean	0.3521	0.3526	0.3522	0.2219
Computation Time Mean (ms)	2645	4692	2496	2442
For $T = 20$:				
Last Position Mean	97.68	124.35	169.00	79.15
Avg. Position Mean	100.24	118.06	137.75	96.32
Any Iteration Top-N Percentage	76.58%	76.19%	76.69%	78.51%
First Iteration Top-N Mean	5.74	5.62	4.54	5.97
Success Percentage	55.77%	48.52%	44.36%	59.95%
Preference Reorder Count Mean	–	–	–	5.8
Property Scan Effort Mean	0.1647	0.1648	0.1648	0.1045
Value Scan Effort Mean	0.4407	0.4413	0.4404	0.2836
Computation Time Mean (ms)	5259	10100	5315	5088

Table 2: Experimental results for $T = \{5, 10, 20\}$, $N = 20$, and $\alpha = 0.9$

difference w.r.t. our approach without user preference ranking ($p < 0.00001$). The reason for this is that the p-norm algorithm is not able to cope with the quantitative properties properly. However, we do observe that the p-norm achieves a higher average for the success percentage than

our approach that includes user preference ranking. The reason for this difference (even though not significant with $p = 0.39608$) might be explained due to the fact that the user in our approach spends 1.1 out of 5 clicks on average on facet re-ordering. This results in less clicks used for the actual query refinement.

However, our approach with user preference ranking does have the best performance on property and value scan effort (with $p < 0.00001$). The reason for this is similar to why the success percentage is relatively low: some user effort is moved from scanning facets to re-ordering facets. We argue that this effort can be more useful for search engines than selecting new facets because it adds more information as to how to rank the products, without making the actual query more complex (i.e., introducing new facets). We also see that our approach (both with and without user preference ranking) outperforms the other algorithms on the 'first iteration top-N' metric (with $p < 0.00001$). We can conclude from this that our approach significantly improves the ability to promote the target product while the user is searching.

When we look at the results for $T = 10$, we notice that the performance of all algorithm improves (e.g., the success percentage increases for all approaches). This is as expected because the queries are more refined when 10 clicks are used instead of 5. We further observe that the 'simple rank' algorithm again outperforms the other approaches on the last position metric ($p < 0.001$) and that our algorithms are again the best w.r.t. the first iteration top-N metric. Another observation that we make is that property and value scan efforts have increased for all approaches. This is due to the fact that the more clicks there are, the more scanning occurs because already selected facets are skipped in subsequent iterations, as facets cannot be selected twice. However, our approach with user preference ranking still outperforms the others (with $p < 0.00001$). The reason for this is that also the number of clicks spent on property scanning is increased (i.e., 2.8 out of 10).

For $T = 20$ we make an interesting observation: compared to $T = 10$, all approaches except our algorithm with user preference ranking perform worse for the last position metric. The reason for this is the increased probability of including incorrect facets in the query, which obviously negatively impact the last position metric. We conclude from this that property re-ordering by the user can combat this issue by placing properties lower on the preference list for which the user is unsure if the target product has them. Furthermore, we see that for the first iteration top-N metric again our approach without the user preference ranking is the best and that similar to previous findings,

the scanning effort for properties and values has increased. However, our approach, without the user preference ranking, does not perform so well for the last position metric. After performing error analysis, we found out that this is caused by the fact that our similarity for qualitative values, which relies on co-occurrence values, does not always help. What we do observe is that our approach that includes user preference ranking seems to resolve this issue, as it scores the best for the last position metric with an average last position of 79.15.

4.3. Results Using an Experiment with Users

Besides the extensive experiments performed using simulation, we also performed an experiment with real users. We had a total of 27 users who participated in the experiment, consisting of 17 males and 10 females. There were 19 users that were between 20 and 30 years old, 6 users that were between 31 and 40 years old, and 2 users that was between 40 and 50 years old. These users were mostly students and colleagues from our university and other universities and there was no financial reimbursement for the participation in the experiment. When selecting these users, we aimed to achieve a balance between males and females in order to reduce bias. Furthermore, we decided to focus on people aged between 20 and 50 years old as these people are more likely to use product search in real life and hence provide a more accurate representation of who would benefit from this algorithm.

The implementation of the Webshop that implements the algorithm proposed in this paper can be found online[1]. We have also implemented the 'standard' Webshop[2], i.e., one that has no special features other than those commonly encountered on the Web. The experiment consisted of 10 small tasks in which each user was given 7 product features and their corresponding preference scores, indicating how important the product features are. The goal was to find the 10 products that best match the given product features, i.e., to find the 10 products that have the highest score according to the preference list. The assigned preference scores were computed using the Zipfian distribution and multiplied with 100 and rounded to the nearest integer (e.g, $1/1 \times 100$, $1/2 \times 100$, $1/3 \times 100$, etc.). The users were performing the tasks using only the 'standard' system. The scores for our system were simply computing by selecting the facets in the order of how they were presented in the task, and answering with the 10 products that were returned by the system.

[1] `http://approx-prod-search.eur.dvic.io`

[2] `http://std-prod-search.eur.dvic.io`

Event type	Total occurrence	Average per user occurrence
Numeric facet change	2142	107.10
Toggle collapsed	389	19.45
List facet select	372	18.60
List facet deselect	135	6.75
Boolean facet change	129	6.45
Numeric facet remove	56	2.8
Boolean facet remove	23	1.15

Table 3: Event counts for the user experiment.

Using this scoring system, our approach achieved an average score of 13.945 while the standard system achieved an average score of 9.170, which we found to be statistically significant using a paired two-sample t-test (with $p = 0.04149$). Table 3 shows the behavior of an average user who participated in the experiment. We can see that most users chose to filter based on numeric facets (such as the price). This also shows us that the users selected on average at least 18 facets to come up with their answers. Using our system this is does not need to be more than 7, i.e., the number of presented product features.

5. Conclusion

In this paper we propose a novel framework specifically geared towards approximate faceted search within a product catalog of a Web shop. It explores the concept of facet similarity functions for both quantitative and qualitative facets, in order to express the degree of similarity between various facets. We propose adaptations to the classical p-norm extended Boolean model to account for the domain-specific characteristics of faceted search in an e-commerce environment. Our approach allows the user to specify a property preference list and takes this into account when weighting the query terms.

In order to assess the performance of the proposed algorithm, it is compared with a simple

baseline algorithm, based on the fraction of facets in the product data that match those in the query, as well as the p-norm extended Boolean model. It is shown that in many cases the proposed algorithm compares favorably to the other algorithms w.r.t. many different metrics (especially for large number of clicks). Our approach also scores best with regards to the percentage of search sessions in which the target product has appeared in the top-N results at least once. Also, it is better able to exploit the quantitative facets in order to promote the target product in the result set. Besides the experiments we performed using evaluation we have also performed an experiment with real users. From this experiment we can conclude that our approach finds more relevant products using less effort.

In future work we would like to explore the usage of ontologies that contain information about similarities between qualitative entities for the purpose of computing a similarity [43]. Another enhancement to the framework could be the inclusion of the negation operator for defining queries. For future work we could also consider using different methods to solve the addressed problem, for example by using the fuzzy set model or semantic search [44].

References

[1] J. B. Horrigan, Online Shopping, Pew Internet & American Life Project Report 36 (2008) 1–32.

[2] Y. Ding, D. Fensel, M. Klein, B. Omelayenko, The Semantic Web: Yet Another Hip?, Data & Knowledge Engineering 41 (2002) 205–227.

[3] H. Zo, K. Ramamurthy, Consumer Selection of E-Commerce Websites in a B2C Environment: A Discrete Decision Choice Model, IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 39 (2009) 819–839.

[4] D. Tunkelang, Faceted Search, Synthesis Lectures on Information Concepts, Retrieval, and Services 1 (2009) 1–80.

[5] M. Hearst, Design Recommendations for Hierarchical Faceted Search Interfaces, in: Proceedings of the ACM SIGIR Workshop on Faceted Search, ACM, 2006, pp. 1–5.

[6] D. Vandic, J. van Dam, F. Frasincar, Faceted Product Search Powered by the Semantic Web, Decision Support Systems 53 (2012) 425–437.

[7] M. Tsagkias, T. H. King, S. Kallumadi, V. Murdock, M. de Rijke, Challenges and Research Opportunities in eCommerce Search and Recommendations, in: ACM Sigir Forum, volume 54, ACM, pp. 1–23.

[8] M. Hearst, A. Elliott, J. English, R. Sinha, K. Swearingen, K.-P. Yee, Finding the Flow in Web Site Search, Communications of the ACM 45 (2002) 42–49.

[9] B. Kules, R. Capra, M. Banta, T. Sierra, What Do Exploratory Searchers Look at in a Faceted Search Interface?,

in: Proceedings of the 9th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL 2009), ACM, 2009, pp. 313–322.

[10] K.-P. Yee, K. Swearingen, K. Li, M. Hearst, Faceted Metadata for Image Search and Browsing, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2003), ACM, pp. 401–408.

[11] J. C. Fagan, Usability Studies of Faceted Browsing: A Literature Review, Information Technology and Libraries 29 (2010) 58.

[12] G. Salton, E. A. Fox, H. Wu, Extended Boolean Information Retrieval, Communications of the ACM 26 (1983) 1022–1036.

[13] Tweakers.net, Pricewatch, `https://tweakers.net/pricewatch/`, 2020.

[14] Q. Liu, E. Chen, H. Xiong, C. H. Ding, J. Chen, Enhancing Collaborative Filtering by User Interest Expansion via Personalized Ranking, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 42 (2012) 218–233.

[15] R. Burke, Interactive Critiquing for Catalog Navigation in E-commerce, Artificial Intelligence Review 18 (2002) 245–267.

[16] P. Pu, L. Chen, Integrating Tradeoff Support in Product Search Tools for E-commerce Sites, in: Proceedings of the 6th ACM Conference on Electronic Commerce (EC 2005), ACM, 2005, pp. 269–278.

[17] B. Li, A. Ghose, P. G. Ipeirotis, Towards a Theory Model for Product Search, in: Proceedings of the 20th International Conference on World Wide Web (WWW 2011), ACM, 2011, pp. 327–336.

[18] V. Sinha, D. R. Karger, Magnet: Supporting Navigation in Semi-structured Data Environments, in: Proceedings of the 24th ACM International Conference on Management of Data (SIGMOID 2005), ACM, 2005, pp. 97–106.

[19] G. M. Sacco, Y. Tzitzikas, Dynamic Taxonomies and Faceted Search: Theory, Practice, and Experience, volume 25, Springer, 2009.

[20] B. Zheng, W. Zhang, X. F. B. Feng, A Survey of Faceted Search, Journal of Web Engineering 12 (2013) 041–064.

[21] D. Vandic, S. Aanen, F. Frasincar, U. Kaymak, Dynamic Facet Ordering for Faceted Product Search Engines, IEEE Transactions on Knowledge and Data Engineering 29 (2017) 1004–1016.

[22] J. L. Herlocker, J. A. Konstan, A. Borchers, J. Riedl, An Algorithmic Framework for Performing Collaborative Filtering, in: Proceedings of the 22nd ACM Annual International Conference on Research and Development in Information Retrieval (SIGIR 1999), ACM, 1999, pp. 230–237.

[23] G. M. Sacco, Y. Tzitzikas, et al., Dynamic Taxonomies and Faceted Search: Theory, Practice, and Experience, volume 25, Springer, 2009.

[24] D. Dash, J. Rao, N. Megiddo, A. Ailamaki, G. Lohman, Dynamic Faceted Search for Discovery-Driven Analysis, in: Proceedings of the 17th ACM Conference on Information and Knowledge Management, ACM, pp. 3–12.

[25] J. Koren, Y. Zhang, X. Liu, Personalized Interactive Faceted Search, in: Proceedings of the 17th International Conference on World Wide Web (WWW 2008), ACM, 2008, pp. 477–486.

[26] D. Vandic, F. Frasincar, U. Kaymak, Facet Selection Algorithms for Web Product Search, in: Proceedings of the 22nd ACM International Conference on Information & Knowledge Management (CIKM 2013), ACM, 2013, pp. 2327–2332.

[27] H.-J. Kim, Y. Zhu, W. Kim, T. Sun, Dynamic Faceted Navigation in Decision Making Using Semantic Web

Technology, Decision Support Systems 61 (2014) 59–68.

[28] X. Niu, X. Fan, T. Zhang, Understanding Faceted Search from Data Science and Human Factor Perspectives, ACM Transactions on Information Systems 37 (2019) 14:1–14:27.

[29] E. Ali, A. Caputo, G. J. Jones, A Comprehensive Survey of Facet Ranking Approaches Used in Faceted Search Systems, Information 14 (2023) 387.

[30] Y. Zhu, D. Jeon, W. Kim, J. Hong, M. Lee, Z. Wen, Y. Cai, The Dynamic Generation of Refining Categories in Ontology-Based Search, in: Second Joint International Semantic Technology Conference, Springer, 2013, pp. 146–158.

[31] M. Arenas, B. Cuenca Grau, E. Evgeny, S. Marciuska, D. Zheleznyakov, Towards Semantic Faceted Search, in: Proceedings of the 23rd International Conference on World Wide Web, ACM, pp. 219–220.

[32] M. Arenas, B. Cuenca Grau, E. Kharlamov, S. Marciuska, D. Zheleznyakov, E. Jimenez-Ruiz, SemFacet: Semantic Faceted Search over Yago, in: Proceedings of the 23rd International Conference on World Wide Web, ACM, pp. 123–126.

[33] A. Stolz, M. Hepp, Adaptive Faceted Search for Product Comparison on the Web of Data, in: Proceedings of the 15th International Conference on Web Engineering (ICWE 2015), volume 9114 of *Lecture Notes in Computer Science*, Springer, pp. 420–429.

[34] B. Cuenca Grau, E. Kharlamov, D. Zheleznyakov, Faceted Search over RDF-based Knowledge Graphs, Journal of Web Semantics 37 (2016).

[35] K. S. Jones, A Statistical Interpretation of Term Specificity and its Application in Retrieval, Journal of Documentation 28 (1972) 11–21.

[36] G. Salton, C. Buckley, Term-weighting Approaches in Automatic Text Retrieval, Information Processing & Management 24 (1988) 513–523.

[37] G. K. Zipf, The Psycho-Biology of Language, Houghton Mifflin, 1935.

[38] G. Salton, A. Wong, C.-S. Yang, A Vector Space Model for Automatic Indexing, Communications of the ACM 18 (1975) 613–620.

[39] P. Pu, B. Faltings, Decision Tradeoff using Example-Critiquing and Constraint Programming, Constraints 9 (2004) 289–310.

[40] S. Liberman, R. Lempel, Approximately Optimal Facet Selection, in: Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC 2012), ACM, 2012, pp. 702–708.

[41] S. Liberman, R. Lempel, Approximately Optimal Facet Value Selection, Science of Computer Programming 94 (2014) 18–31.

[42] AWS, Amazon Web Services. Large Cloud Computing Provider offered by Amazon.com, `http://aws.amazon.com`, 2020.

[43] P. D. H. Zadeh, M. Reformat, Assessment of Semantic Similarity of Concepts Defined in Ontology, Infromation Sciences 250 (2013) 21–39.

[44] A. M. Mahdi, A. S. Hadi, The Current State of Linked Data-based Recommender Systems, in: 2nd International Conference of Information Technology to Enhance E-learning and Other Application (IT-ELA), IEEE, pp. 154–160.