Personalized Keyword Search with Partial-Order Preferences

Achim Leubner, Werner Kießling
Institute of Computer Science
University of Augsburg, Germany.
{leubner, kiessling}@informatik.uni-augsburg.de

Abstract

Personalized search engines must be able to cope with various user preferences, retrieving the best matches to a query. For SQL and XML applications new methods for such preference-based searches have been implemented recently. Here we adopt this approach to keyword search in full-text search engines. We propose to augment the vector space model (VSM) by preference constructors having an intuitive partial order semantics: Pareto-accumulation and prioritization. We show that prioritization can be interpreted as subspace preference in the VSM. Using a preprocessor approach we succeed to map prioritization onto the VSM. A first query benchmark, using the standard Time-collection, revealed promising results. The retrieval quality, measured by average expected search length, could be slightly improved. Using the proposed meta engine approach, this gain in retrieval quality is accompanied by a substantial speed up of query runtimes. Thus subspace preferences can be integrated efficiently into existing full-text search engines.

1 Introduction

Personalization of Internet services and of search engines has many facets. In customized systems the user has to select between different options himself ([16]). A further step towards personalization is that the system automatically adapts the presented information and the form of the presentation, depending on the actions of a user, his personal preferences and profile or perhaps also the profiles of similar users. A good example are online newspapers. Typically, instead of subscribing to a whole newspaper, a user might select categories like politics, local news, sports, etc., i.e. he can customize the newspaper. However, with existing information retrieval techniques, online newspapers could be much more intelligent. They could filter the information in a far more personalized way (e.g. [15]). But many full-text retrieval models can be seen either as too simple or too complex for personalization issues. The simple ones like coordination level matching typically lack a sufficient capability to capture the semantics of the users information needs and therefore deliver poor retrieval results. On the other hand there are models like boolean retrieval, which are complex enough to formulate the “ideal” query delivering only and all of the relevant documents, if the available predicates are powerful enough. Unfortunately these queries are not feasible in practice: The user needs very detailed information about the documents to formulate the queries. But this information is only available after identification of all relevant documents ([4]) which is the reason to ask the query. Correspondingly boolean retrieval delivers poor results ([23], [22], [19], [4]).

In this paper we want to propose an intuitive retrieval model, which is compatible to the commonly used vector space model (VSM) and enables an easy formulation of semantically richer queries to improve the personalized retrieval quality. For multi-attribute search engines recently a model for preferences has been developed that is formally founded on strict partial orders and at the same time provides an intuitive semantics for the user ([7], [3]). This model provides a set of base preferences together with a choice of complex preference constructors to inductively engineer preferences that
can be tailored to the user’s personal wishes as closely as desired. Two salient preference constructors
are the so-called Pareto-accumulation, which combines partial orders on an equally important basis,
and the so-called prioritization, which treats the partial orders with decreasing importance. This
preference model has been the foundation to extend SQL and the query language XPATH for XML
by soft constraints ([11], [7], [10], [8]). Such preference-based queries return the best matches,
supporting a “k-best” query model. This approach has already been implemented as a commercial
product for SQL-based search engines and deployed in various portal applications in the B2C e-
shopping market ([9]). Encouraged by good experiences in those domains we aim at adopting this
approach to simple keyword search in information retrieval. Typing a set of keywords into a simple
search mask is familiar to most users of Internet search engines or of most e-shops or mobile e-shops.
But as most Internet users have experienced by themselves, those simple keyword searches often
exhibit an inferior retrieval quality. Keyword search is definitively convenient, provided the proper
keywords come into one’s mind. However, if intuitive features for better personalization would be
provided, then this would certainly be welcomed a lot. In this spirit we want to show an evolutionary
way to upgrade existing keyword-based search. Importantly, we will stick to the established VSM as
a proven basis to calculate a numeric score for each keyword, applying whatever suitable indexing
scheme. On top of that we are going to offer Pareto-accumulation and prioritization as an additional
means for the user to express personal preferences.

The rest of this paper is organized as follows: Subspace preferences will be described in detail in
section 2. Section 3 discusses selected implementation aspects. First benchmark results regarding the
quality of the retrieval results and the runtimes for our prototype are given in section 4. It is followed
by a section about related work. Finally, section 6 gives a short summary and outlook.

2 Subspace Preferences

The idea behind subspace preferences is giving the user the possibility to tell, if search terms are
equally or more important than others. According to [6] in the first days of the VSM, the user was
allowed to do so by assigning weights to the search terms, but subsequent studies showed that au-
tomatically calculated weights perform better. One reason may be that word distributions should be
taken into account, which cannot be done by the user with reasonable effort. Nonetheless the user can
divide the search terms into subsets and tell which subsets are equally and which are more important.
More technically, the search term subsets form subspaces of the VSM. In this paper we assume a
standard VSM with cosine similarity measure as described in [6]. We also assume that cosine nor-
malization is done during indexing, i.e. the documents scores are calculated as scalar products. The
subspaces are then calculated as VSM’s of their own.

Let’s consider the case of 2 subspaces and let \( \langle D, \geq_{P_1} \rangle \) and \( \langle D, \geq_{P_2} \rangle \) denote the partial orderings
of the “sub-vector space models”. We define:

\[
D_1 > D_2 \iff \text{score}(D_1) > \text{score}(D_2)
\]

In the next step the results of each subspace can be combined according to the user’s wishes by
Pareto-accumulation and prioritization.

Pareto-accumulation (\( \otimes \)): Pareto-accumulation of two partial orders \( P_1 = \langle D, \geq_{P_1} \rangle \) and \( P_2 = \langle D, \geq_{P_2} \rangle \)
on the set of documents \( D \) can be defined for all \( D_1, D_2 \in D \) as

\[
D_1 >_{P_1} \otimes_{P_2} D_2 \iff [D_1 >_{P_1} D_2 \land (D_1 >_{P_2} D_2 \lor D_1 =_{P_2} D_2)] \\
\lor[D_1 >_{P_1} D_2 \land (D_1 >_{P_1} D_2 \land D_1 =_{P_2} D_2)]
\]

Document \( D_1 \) is better than document \( D_2 \), iff \( D_1 \) is really better than \( D_2 \) in at least one sub-
space and \( D_1 \) is at least as good as \( D_2 \) with respect to all other subspaces. Thus semantically
the combined subspaces are equally important. Pareto-accumulation corresponds to the product
or coordinate-wise order. If the indexing within the subspaces is formed by a scoring function as assumed by (*) above, it is also known as SKYLINE operator ([2]).

**Prioritization (&):** Prioritization of two partial orders \( P_1 = \langle D, \geq_{p_1} \rangle \) and \( P_2 = \langle D, \geq_{p_2} \rangle \) on the set of documents \( D \) can be defined for all \( D_1, D_2 \in D \) as

\[
D_1 >_{P_1 \& P_2} D_2 \iff D_1 >_{P_1} D_2 \lor (D_1 = D_2 \land D_1 >_{P_2} D_2)
\]

A document is better than another document, iff it’s better according to the most important subspace. If it’s equal there, the next subspace decides. Prioritization corresponds to the lexicographic order. Above definitions can be generalized to \( n \) preferences. Pareto-accumulation and prioritization are known to be partial orders again ([7]).

**Example 1 Vector Space Model**

Our sample user Willy has an equal interest in jazz and classic, preferably from England. Confronted with a simple keyword search engine, Willy’s only option is to enter the 3 keywords “[jazz, classic, England]”. Using a VSM, assume that this query will be indexed as in Table 1. In the VSM, the documents scores have then to be calculated as scalar products of the document vectors with the query vector. Finally, the documents are sorted according to this score which gives the result in Table 2.

Here we have the frequent case that no perfectly matching documents can be found, instead best-matching alternatives must be supplied. But note that apparently document \( D_4 \) is not about music, which is in contradiction to Willy’s interest. It is rated as best document only due to its high weight for “England”.

<table>
<thead>
<tr>
<th>Search Term</th>
<th>Query</th>
<th>Document 1</th>
<th>Document 2</th>
<th>Document 3</th>
<th>Document 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jazz</td>
<td>1.00</td>
<td>0.1</td>
<td>0.2</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Classic</td>
<td>0.33</td>
<td>0.9</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>England</td>
<td>0.51</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.8</td>
</tr>
</tbody>
</table>

The obvious solution to the wrong position of document \( D_4 \) would be a decrease of the weight of “England” in the query vector. But it’s a bad idea to give a user direct control over this values. A more practical way is the use of subspace preferences.

**Example 2 Subspace Preference**

Now assume that Willy has access to a keyword search that supports the prioritization operator ‘\&’ for example by an appropriate GUI. Thus the fact that the musical aspects “Jazz” and “Classic” are more important to him than “England” leads to the query “[Jazz, Classic] & [England]” in figure 1, consisting of two subspaces, one about music and one with Willy’s geographic preference. Both subspaces will be combined by prioritization according to Willy’s wish. Please note that the query weights remain the same as do the index values.

<table>
<thead>
<tr>
<th>Search Term</th>
<th>Query</th>
<th>&amp;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jazz</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>Classic</td>
<td>0.33</td>
<td></td>
</tr>
<tr>
<td>England</td>
<td>0.51</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\)We assume a standard tf-idf index function with cosine normalization.
Now the system has to calculate the scores of the subspaces as scalar product:

Table 3: Partial results with subspace preference I.

<table>
<thead>
<tr>
<th>Document</th>
<th>Score</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Jazz, Classic]</td>
<td>[England]</td>
<td></td>
</tr>
<tr>
<td>$D_1$</td>
<td>0.40</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>$D_3$</td>
<td>0.20</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>$D_2$</td>
<td>0.10</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>$D_4$</td>
<td>0.00</td>
<td>0.41</td>
<td></td>
</tr>
</tbody>
</table>

In this specific example, the first subspace already suffices and gives the resulting order. Willy could express his wishes in an easy and intuitive way, i.e. without calculation of weights or word distributions. This way he is able to include more semantics which leads to the desired result.

**Example 3 Subspace Preference II**

Finally assume the keyword search engine supports both '&’ and Pareto-accumulation ‘$\otimes$’. Then Willy’s intention can be represented as “([Jazz] $\otimes$ [Classic]) & [England]”. The calculation of the three subspaces of this query leads to the partial results in table 4.

If we now compare the documents to each other, then for example document $D_1$ is better than document $D_2$, since it is really better with respect to “Classic” and equally good with respect to “Jazz”. Document $D_3$ and $D_5$ are incomparable, because each is really better in one of the two cumulated subspaces.

Table 4: Partial results with subspace preference II.

<table>
<thead>
<tr>
<th>Document</th>
<th>Score</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Jazz]</td>
<td>[Classic]</td>
<td>[England]</td>
</tr>
<tr>
<td>$D_1$</td>
<td>0.10</td>
<td>0.30</td>
<td>0.00</td>
</tr>
<tr>
<td>$D_2$</td>
<td>0.10</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>$D_3$</td>
<td>0.20</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>$D_4$</td>
<td>0.00</td>
<td>0.00</td>
<td>0.41</td>
</tr>
</tbody>
</table>

Again, the geographic subspace does not contribute to the result in this example. The result order of this query is $D_1 > D_2; D_3 > D_2; D_2 > D_4$. We see that in this case the desired correction of the vector space result can be achieved too.

3 Implementation Aspects

3.1 Implementation Architectures

The preference may be built up in several steps: In the first step, the user specifies search terms. Based on a thesaurus, user profiles etc., the system may then suggest subspaces and the combination of them. The user can accept or reject the subspaces and specify, which ones are more or equally important. Finally, the system may append sub-preferences like “sort by length” based on the user’s profile.

Subspace preferences can be implemented straightforward using a “meta engine approach” as shown in figure 2. The subspaces are evaluated by a standard vector space engine. Only the analysis and decomposition of the query as well as the combination of the partial results is done in a separate combination component. This way both prioritization and Pareto-accumulation can be implemented. On the other hand, this implementation can get quite complex, if additional attribute-based preferences are allowed. Details about the implementation of attribute-based preferences are described in [2] and [20]. Subsequently we will describe a transformation of prioritized VSM’s onto a single
With this transformation subspace preferences may also be implemented via a preprocessor approach as shown in figure 3.

A preprocessor maps the prioritized query onto a vector space query, which is evaluated via a conventional vector space engine. The result of the vector space engine can then be presented to the user. Unfortunately, with this implementation only prioritization can be used: Pareto-accumulation of (totally ordered) partial results can be partially ordered. But the results of VSM’s are always totally ordered. So a general transformation of cumulated subspaces onto a VSM is not possible. Of course, a combination of both approaches is possible. An implementation could for example transform all prioritizations of subspace models and suitable attribute-based preferences onto a subspace model and combine the partial results separately.

3.2 Preprocessor Approach for Prioritization

We now present a transformation of prioritized VSM’s onto one VSM. The idea is to increase the weights of more important search terms so much that even the slightest possible differences concerning these terms surpass the maximal possible values of less important terms. That means we construct a query vector in the complete vector space, which is composed by the query vectors of the subspaces multiplied by appropriate factors. The factor of the $i$-th subspace is calculated as $k_{a}$ where $k_{a}$ is the maximal possible score for the $n$-th subspace and $k_{c}$ is set to $o$.

This leads us to the following formula for the total score of document $D_i$ for all $k$ subspaces:

$$b_i = \sum_{i=1,\ldots,k} \sum_{e_i} [\prod_{j=0,\ldots,i-1} m_j + 1] \cdot r_Q(e_i) \cdot r_D(e_i)$$

I.e. for every subspace $i$ we add for all search terms $e_i$ of that subspace the product of the subspaces factor with the weights of the query and document vectors $r_Q(e_i)$ and $r_D(e_i)$.

Now the expression $\star$ can be seen as a new query vector $r'_Q$. W.l.o.g. we assume that all subspaces are disjoint. If a search term is used in several subspaces, the different occurrences have to be handled like distinct terms. Of course, the corresponding values in the document vectors and with it the indices remain the same. With this assumption we are able to sum up the subspaces to a new vector space,

---

2 This nice transformation property is an important reason why we use the VSM as basis. Of course, if we don’t need this property, it is also conceivable to use other retrieval models for the partial preferences.

3 A more detailed discussion of this transformation can be found in [13].

185
which is built by all search terms. This way we get the following formula:

\[ b_i = \sum_e r_Q^i(e) \cdot r_D^i(e). \]

\( r_Q^i \) is a combination of the modified query vectors \( r_Q^i \) and \( r_D^i \), a combination of the document vectors \( r_D^i \). Since finite total orderings can be mapped onto \( (N_0, \leq) \), they can be seen as one-dimensional VSM:

\[ r_D = \sum_e 1 \cdot r_D(e) \]

The search term \( e \) has the purpose to mark the one-dimensional vector space. \( r_D \) does not really depend on it. This way it is possible to integrate arbitrary finite total preferences into a VSM using the preprocessor approach we described.

**Example 4** We reconsider Willy’s query from example 2, look for jazz or classic music prior to England. Due to his time restrictions, Willy prefers to see the shorter documents of equally scored ones first. The query now reads as “[Jazz, Classic] & [England] & sort(length)” which leads to the following subspaces:

**Table 5: Willy’s query from example 2 reconsidered.**

<table>
<thead>
<tr>
<th>Document</th>
<th>Subspace 1</th>
<th>Subspace 2</th>
<th>Subspace 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Jazz</td>
<td>Classic</td>
<td>England</td>
</tr>
<tr>
<td>( D_1 )</td>
<td>10</td>
<td>90</td>
<td>0</td>
</tr>
<tr>
<td>( D_2 )</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( D_3 )</td>
<td>20</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( D_4 )</td>
<td>0</td>
<td>0</td>
<td>80</td>
</tr>
<tr>
<td>Query</td>
<td>100</td>
<td>33</td>
<td>51</td>
</tr>
</tbody>
</table>

The first subspace of Willy’s query is about his stylistic wishes. The second, less important subspace is about Willy’s geographic preference for England. The last, least important preference is a natural ordering on the length of the documents, which is also appended by prioritization. To avoid problems with the range of integer types, we reduce the maximal value of this preference, i.e. we normalize it onto the range \( [0, 1000] \). This means, we only roughly sort equally scored documents by their length. Then we subtract the normalized value from 1000, to get higher values for shorter and therefore better documents. The maximal possible value for this preference is 1000.

**Table 6: Modified query vector.**

<table>
<thead>
<tr>
<th></th>
<th>( r_Q^i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jazz</td>
<td>100 \cdot 101101 = 10110100</td>
</tr>
<tr>
<td>Classic</td>
<td>33 \cdot 101101 = 3336333</td>
</tr>
<tr>
<td>England</td>
<td>51 \cdot 1001 = 51051</td>
</tr>
<tr>
<td>Mod. length</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 7: Result for the preprocessor approach.**

<table>
<thead>
<tr>
<th>Document</th>
<th>( b_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( D_1 )</td>
<td>401371220</td>
</tr>
<tr>
<td>( D_2 )</td>
<td>101101300</td>
</tr>
<tr>
<td>( D_3 )</td>
<td>303303100</td>
</tr>
<tr>
<td>( D_4 )</td>
<td>4084080</td>
</tr>
</tbody>
</table>

The multiplication factors are 1 for the length preference, 1000 + 1 for the geographic subspace and \((1000 + 1) \cdot 100 + 1000 + 1 = 101101\) for the stylistic subspace. With these values we can now calculate the modified query vectors \( r_Q^i \). In this example, we immediately combine them to the query vector \( r_Q^i \). Since the subspaces are disjoint, we don’t need any corrections here. The document vectors \( r_D^i \) in the combined VSM correspond directly to the document vectors \( r_D^i \). Their values always remain the same. Now we can calculate the total score in the combined VSM as scalar product \( b_1 = \sum_e r_Q^i(e) \cdot r_D^i(e) \). This way we get the same correct ordering as in example 2: \( D_1 > D_3 > D_2 > D_4 \).
The above example shows that prioritization can be seen as a technique to adopt the weights of the VSM semi-automatically without the cognitive overhead of changing the weights themselves by hand. It is also easy to recognize that the total score can become very large, depending on the number and the range of the preferences combined. Normal VSM engines will typically use integer or float types with a too small range, so overflows are likely to occur. An alternative might be to transform only some subspaces onto a single VSM and to combine the results of these VSM’s separately. The extreme case of this principle corresponds to the “meta engine” approach in figure 2.

4 Prototype Evaluation

4.1 Test Environment

Our choice of a first test-bed was guided by the following considerations. The very nature of the preference operators ‘∅’ and ‘&’ and the experiences gathered for multi-attribute search in SQL and XML environments suggested to us that a benchmark on document collections characterizing categorical data should produce favorable results to our approach. Such collections are typical for e-shopping applications, e.g. for e-portals for job search, real estate and housing, dating services or used car sites, just to name a few. However, to our knowledge no standard benchmarks exist for these domains, which is indispensable to arrive at objective conclusions concerning the retrieval quality. Therefore we decided to pick a well-known benchmark for a collection of complex documents of political articles. This complex terrain can of course be considered as a sort of worst-case test for our approach.

For this preliminary test we decided on the Time collection⁵, arbitrarily taking the first 10 queries. Each query has been run in three ways on a prototype implementation written in Java with an architecture similar to the meta engine approach described above:

1. As a vector space query, indexed by Smart⁶.
2. As a semantically motivated subspace preference.
3. As a subspace preference, where arbitrary subspaces and combinations of the partial results are allowed, just to get the best possible results (“optimal queries”).

The idea behind the semantically motivated queries was the question: “Is it good to partition the vector space based on semantic considerations?” If so, it may be supported using tools like thesauri or ontologies. For the “optimal queries”, the idea was to test how good the results can be and to see, if there may be a structure behind these queries. Of course, the search space was too large to test all possible queries, so we only tried to reach local optimas.

We will give a short example of our tests. Consider e.g. the 3rd query of the Time-collection:

“NUMBER OF TROOPS THE UNITED STATES HAS STATIONED IN SOUTH VIET NAM AS COMPARED WITH THE NUMBER OF TROOPS IT HAS STATIONED IN WEST GERMANY.”

Smart creates the following query vector:

<table>
<thead>
<tr>
<th>Search Term</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>west</td>
<td>0.172390</td>
</tr>
<tr>
<td>troop</td>
<td>0.336340</td>
</tr>
<tr>
<td>stat</td>
<td>0.146920</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Search Term</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>numb</td>
<td>0.489690</td>
</tr>
<tr>
<td>german</td>
<td>0.270150</td>
</tr>
<tr>
<td>compar</td>
<td>0.443560</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Search Term</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>nam</td>
<td>0.355720</td>
</tr>
<tr>
<td>station</td>
<td>0.355720</td>
</tr>
<tr>
<td>unit</td>
<td>0.270150</td>
</tr>
</tbody>
</table>

⁵The Time collection is available from ftp://ftp.cs.cornell.edu/pub/smart and http://www.dcs.gla.ac.uk/idsom/ir_resources/testcollections/
⁶We used the version of Smart available from ftp://ftp.cs.cornell.edu/pub/smart/.
To create a sample semantically motivated subspace preference, we first built the semantic subspaces and a semantically justified combination of them. For query 3 we claim, the subspace [troop] is most important. Then, we chose the subspaces [west, german] and [nam] as the second stage. Both have equal rights, so they are combined by cumulation. [unit, stat] was chosen as third stage, since there are only two countries, the United States and France, which had troops in West Germany and Viet Nam, so [west, german] and [nam] seem to be more important in this query. The least important subspace collects all the remaining dimensions. The resulting semantically motivated subspace preference is:

\[[\text{troop}] \& ( [\text{west, german}] \otimes [\text{nam}] ) \& [\text{unit, stat}] \& [\text{compar, numb, station}]\]

For the “optimal” subspace preference any combination of arbitrarily built subspaces is allowed, as long as it delivers the best results. For query 3, the “optimal” subspace preference is:

\[[\text{troop, german, nam}] \& [\text{unit, stat}]\]

Since there are quite some critical points concerning the customary retrieval measures precision and recall ([18], [17]), we use as retrieval measure the Average Expected Search Length (ESL) as in [12] extended for partial orders. It states, how many documents need to be retrieved, until the next relevant document is reached. Of course, the optimal ESL-value is 1. The next document the user looks at is relevant, until all relevant documents have been seen. If there are multiple documents with the same score, all possible orderings are taken into account.

To extend the average ESL to partial orders, we assumed that the user looks through the result “breadth first”, i.e. he looks through the whole level before he takes the next one into account and that the order in a level is random. Therefore we regarded each level as a class of equally scored documents.

4.2 Retrieval Quality

Before jumping to conclusion we must explain an important correlation between the subspaces of the VSM, applying a numerical ranking procedure, and the preference operator ’&’, modeling ordered importance between subspaces. According to its definition ’&’ does not regard less important subspaces if there is a distinction in the more important subspace. Now note that such a rigid policy is only justified, if the calculations of numerical scores for the subspaces are perfectly correct. In the event of categorical data and multi-attribute search, this prerequisite is normally met. However, this does not hold in general for the indexing methods used in the VSM to calculate the document scores. As an immediate consequence, it becomes obvious that too much precision may be harmful in the following sense: A document $D_1$, which is only marginally better than $D_2$ in its first subspace, will always dominate over $D_2$ regardless of the scores of $D_1$ and $D_2$ for the second subspace. Thus, being aware that document ranking in the VSM is only an approximation of relevance, we must aim to remedy this situation by making the decision for “$D_1 > D_2$” more error-tolerant.

One possibility to implement such a strategy is to deliberately reduce the precision of the underlying VSM engine. Indeed, our tests verified the conjectured behavior. The experiments showed that a precision of 2 leads to good results for the average ESL-values. The VSM itself has been evaluated with a high precision of 6 like in Smart (VSM/6), since a precision of 2 leads to worse retrieval results in the pure VSM ([13]). As an additional positive side effect this reduction of precision during the evaluation of the subspace preferences is advantageous for the preprocessor implementation, since the factors grow slower (of course, if we implement the meta-approach, we have to reduce the precision in the combining component).
If we look at the values of the average ESL in figure 4 and table 9, we see that the semantically motivated subspace preferences (SSP/sem) perform on average 16% better than the VSM. Unfortunately this is largely due to the results for query 3. But even if we remove the outlier, subspace preferences perform slightly better than the VSM. The “optimal” subspace preferences (SSP/opt) always perform better than the VSM. The only exception is query 8, where the VSM already reaches the best possible value of 1.00. On average the “optimal” queries perform a remarkable 25% better than the VSM giving much room for future improvements.

4.3 Runtime Performance

For the evaluation of the runtimes, we used a prototype implemented in Java. We run the tests on a standard PC, with a Pentium III/550 MHz under Linux, using the IBM-JDK Version 1.3.0. The prototype can be used to calculate different models by replacing the function comparing two documents. It has no data structures, which were specially optimized for a certain retrieval model. This way we were able to directly compare the runtimes of the VSM and of the subspace preferences, since the only algorithmic differences concerned the methods comparing two documents.

On this prototype we evaluated the running times of the first 10 queries of the Time-collection for the three discussed types of queries (VSM/6, SSP/sem/2, SSP/opt/2). We always calculated the whole result order, although a user typically only pays attention to some of the k-best results. This of course bears potential for further improvement of the running times.
Figure 5: Running times

The results of these tests can be seen in Figure 5 and Table 10. In our evaluations, subspace preferences are always substantially faster than the VSM, since in many cases only a few of the subspaces have to be calculated to get the total results, which is faster than the calculation of the whole vector space. Another point is the calculation precision. The submodels of subspace preferences are evaluated with a precision of 2 to give the following stages a chance, the VSM with a precision of 6, since a precision of 2 leads to worse results in the pure VSM ([13]). This leads to a clear advantage for subspace preferences.

5 Related Work

5.1 Partial Orders in Information Retrieval

The term “preference” in the context of this work is synonymous with “strict partial ordering”. Partial orderings can be seen as directed acyclic graphs (DAG’s) where the edges bear a semantic like “node A is better than node B”. In information retrieval there are some considerations, which lead to partial orderings, i.e. to preferences. For example Bookstein claims in [1] that “is better than” relationships have intuitive semantics. Wishes are often presented as “I like this more than that”. He also states “while detailed evaluations of items are often difficult for judges, they seem better able to make preference decisions between pairs of items.”

In [5] it is similarly claimed that today’s information retrieval systems cannot handle relationships like “A is better than B” adequately, although they make sense in many cases and can be obtained quite easily. The authors therefore claim that using this additional information is advantageous. So they treat the rankings of various search engines as “preference judgments”, i.e. as votings that between the documents an “is better than” relationship exists: “Notice that each user’s movie rating can be viewed as a set of preference judgments. In fact, interpreting ratings as preferences is advantageous in several ways: for instance, it is not necessary to assume that a rating of “7” means the same thing to every user.” Cohen, Schapire and Singer miss partial orders only due to the point that different search engines can return different orders, i.e. in the presentation as directed graph, there can be cycles, which are not allowed in preferences. If we suppose that the “better than” relationships model the wishes of one user, there should be no such case, where a user likes one thing really better than another and vice versa. So in this case, we get strict partial orderings, i.e. preferences.

A third point to mention is the fact that “is better than” judgments about documents are also customary in information retrieval as relevance feedback ([24]).
5.2 Multiset Preferences

In [14] a full-text preference based on a multiset-ordering is described. Basically, an excess of occurrences in a “better” search term can be propagated to all “worse” search terms, i.e. an additional occurrence of a better term out-weights all occurrences of worse terms. Which term is “better” or “worse” is given by the user in form of a partially ordered set of search terms.

We analyzed a refined version of this preference, which we call weighted set preference. Our refinement basically transforms the multi-set into a set and weights, associated to the sets’ elements. This way we are more flexible, since we are able to represent occurrences by corresponding weights but are not limited to do so. In fact, we typically used weights calculated by a standard tf-idf index function. We also allowed weights between the terms of the partially ordered set of query terms. It can be shown that the multi-set preference is a special case of the weighted set preference ([13]). But our tests with full-text preferences based on the multi-set order were not successful. In fact, conventional information retrieval models perform better and do so faster ([13]).

6 Summary and Outlook

We have presented an evolutionary approach towards personalization of keyword search under the VSM by two types of intuitive preference operators, Pareto-accumulation and prioritization. We investigated in more detail the latter one, which allows us to partition the vector space into subspaces forming their own VSM. Interfacing prioritization with numerically ranked keyword search required some careful design decisions concerning the precision of the score calculation in the subspaces. In our prototype implementation we used the Smart system as a basis to calculate keyword scores. A benchmark performed with this prototype, using the standard Time-collection for complex document retrieval, showed very encouraging results. For personalized queries from the Time-collection we could observe improvements of the average expected search length ESL over the pure VSM. Since prioritization performs best on categorical data we conjecture that even more significant improvements will be feasible in such application domains.

In summary we have given evidence that personalization by strict partial-order preferences can be very helpful to improve conventional keyword search engines, just as it has been the case with multi-attribute search in SQL and XML. This holds for general Internet search engines, but to an even higher extent for keyword engines in e-commerce. In particular search engines for mobile e-commerce could benefit substantially from a shortened ESL. Due to its subspace property prioritization can be integrated right away into existing keyword search engines, e.g. in XXL ([21]) or even in full-text cartridges for Oracle 9i or extenders for IBM DB2. Pareto-accumulation in turn requires a meta-engine approach, where known efficient algorithms from [2] and [20] can be employed. As a further next step we need to extend our benchmarks to categorical-rich data collections and on complex query benchmarks like TREC (http://trec.nist.gov). Also it is an open issue how prioritization and Pareto accumulation can be integrated into a search mask without impacting the ease of use of simple keyword search. We believe that in a sophisticated personalized search engine in many cases this can be taken over by the system which automatically expands the user query based on semantic knowledge coming from the user’s profile, from preference mining and from domain knowledge extracted from ontologies. Research along these lines is currently in progress.
References


