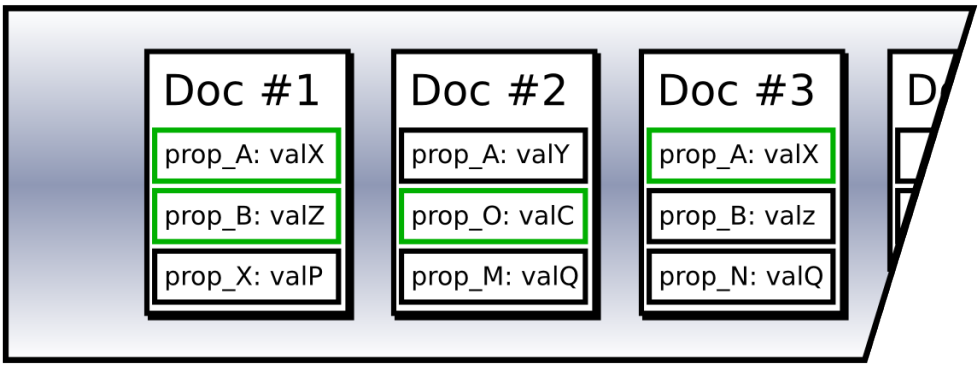
CHALMERS



Faceted search with a large amount of properties

Exploring opportunities for faceted search for an e-commerce application with a large amount of dynamic properties

Master of Intelligent Systems Design thesis within Findability and Human Computer Interaction

Per Fredelius

Department of Applied IT
Chalmers University of Technology
Gothenburg, Sweden 2013
Report No. 2013:014
ISSN: 1651-4769

Contents

1 Abstract

Qalixa.com is a online e-commerce meta search engine that consolidate articles from a growing number of retailers from most lines of retail business. The articles in store entail a variety of meta data types such as properties, tags, category structures and free text.

In this project, the author try to find ways to exploit the meta data and facilities available through a modern Enterprise search solution to create suggestions on how to build a User interface with sufficient competence to deal with this complex search space in an empowering manner, yet represent a simple enough experience for the end user to allow overview as well as a simple as possible learning curve.

A few concepts are prototyped and a variety of options of future work are analyzed. Characteristic issues with applying faceted search to the particular case are found and analyzed to some detail. Concepts that are tested to some extent are *selectable facet filters*, *grouping of properties with clustering*, *fuzzy logic in facet search* and *implicit ordering of documents given filter order*.

1.1 Keywords

Faceted search, Enterprise Search, Dynamic properties, Human-Computer Interaction and Information Retrieval

2 Introduction

Qalixa is a small start up company that drives the development of a search engine hosted at qalixa.com. Qalixa aims in this engine to deliver a free-to-post advertisement service, both for the business to business and for the business to consumer segment.

The purpose of this thesis is to find ways to bring further insight into the particular case that Qalixa represent in the scope of faceted search and user interface design for faceted search, and if possible present opportunities for improvement for Qalixa and for users of such technology in general.

This is realized by analyzing the most prominent issues with the current solution, proposing a set of concepts to alleviate these problems, implementing prototypes for these suggestions and finally evaluate these solutions to bring further insight into the challenges of the case.

2.1 Prototypes

The first proposed solution is that of *selectable facet filters*. It connotes presenting the user with a set of properties recurring in the set of results currently retrieved, and allow the user to select from these properties and in so turning the property into a *filter*. Filters are presented in a separate list from the properties. In such, the user is allowed to manage preferences of interest separately from a larger set of properties that are not of immediate interest. The filters can be *expanded* to present a set of applicable filter values for the user, allowing him/her to select a span.

Secondly, to deal with cases where the number of applicable properties are large, methods for arranging such properties in a logical way are needed. To supply structure properties were *arranged in groups using a clustering algorithm.*

In order to avoid false negatives in a high entropy and heterogeneous data base, it is desirable to have filters that does not work in a inclusive/exclusive but promoting/demoting manner. This method can be compared to the idea of fuzzy logic and is in this thesis referred to as *fuzzy facets*. In that way, articles that happen to lack definitions on particular properties that would otherwise be of the desired kind can be found in the list, albeit further down in the list. In the prototype and in this thesis a simple method for achieving this behavior is suggested.

Another feature is that of *implicit ordering of documents given filter order*. Given a moderate amount of interesting articles that the user wants to compare in detail, being able to compare properties closely should make it desirable to rearrange the order of articles based on values of properties. In this thesis a mean for arranging documents based on the order of filters in the filter list is proposed.

2.2 Questions

- How can faceted search be exploited for cases where dynamic properties are prevailing?
- How can Solr or similar tools be used in a Qalixa?

2.3 Tasks faced

- Production of several GUI concepts for enterprise search and for the e-commerce domain
- Evaluation of search frameworks

- Configuration of Solr for enterprise search
- Development of plug-in for optimization of MySql to Solr data migration
- Implementation of e-commerce search GUI prototype with focus on faceted search
- Analysis of problems with the prototype and conclusions taken
- Implementation of a clustering algorithm
- Formalization of a theoretical model for user centric search in the information retrieval domain

2.4 Outline

This thesis describe the project by first describing the case being studied in chapter 3 after which some relevant methodology and tools for the domain are described in chapter 4. Then, in chapter 5, some concepts where thought up taking inspiration from challenges and opportunities observed. Some of these concepts are turned into a working prototype, that is described in chapter 6. The working prototype is then evaluated and problematized in chapter 7. Lastly, the problems are discussed further and some ways of tackling the issues are suggested in chapter 8.

3 Background

The project consisted of literature studies, concept creation, configuration and implementation of a prototype and finally analysis of the implementation and prospective future work, where a concrete list of issues of the domain could be produced. In the literature studies, the author tried to find potential ways to tackle perceived technical problems of the case in question, search at Qalixa.com. Theoretical studies was made in part as a prelude to the implementation task and in partly after the task, to make use of the lessons learned.

From the outlook it was explicitly of interest to test new search framework technologies. The scope of this project start out from the premise of applying one such technology on the existing database of Qalixa. Another interest was to try out interesting features of this framework to evaluate the opportunity it can bring.

Furthermore, as an academic project, it was of some interest to try to find a research potential. And for the authors sake, to try out interesting technology for the sake of learning.

While user studies could have brought some additional insight and lessons from the prototype, it was outside the scope for the project due to time and resource constraints. It was also deemed that using such method would be premature due to problems with the user experience stemming from the domain problems identified.

Additionally, little data is available from current user behavior. Facilities for analyzing such is not yet in place as the Qalixa venture is still a young one.

In future works, user studies on a prototype using coherent input data can be used for adjusting characteristics of the user interface. And to give an idea of the understandability of the proposed features. Given well designed user tests; it could be concluded whether exotic features actually shorten the time for task completions and how likely such features are to be discovered and understood spontaneously.

Case study: Qalixa Looking at the Qalixa use case a few characteristic opportunities were found. Technical opportunities that are perceived to need a lot of thought in order to leverage the value of the web service. Applying faceted search onto Qalixa present a certain challenge; the number of applicable facet properties can vary greatly between queries, as would show by analyzing the prototype. The total number of properties spanning all articles is unbound as each article can define and carry an arbitrary amount. The relevance and usefulness of any particular property is unknown from the outset.

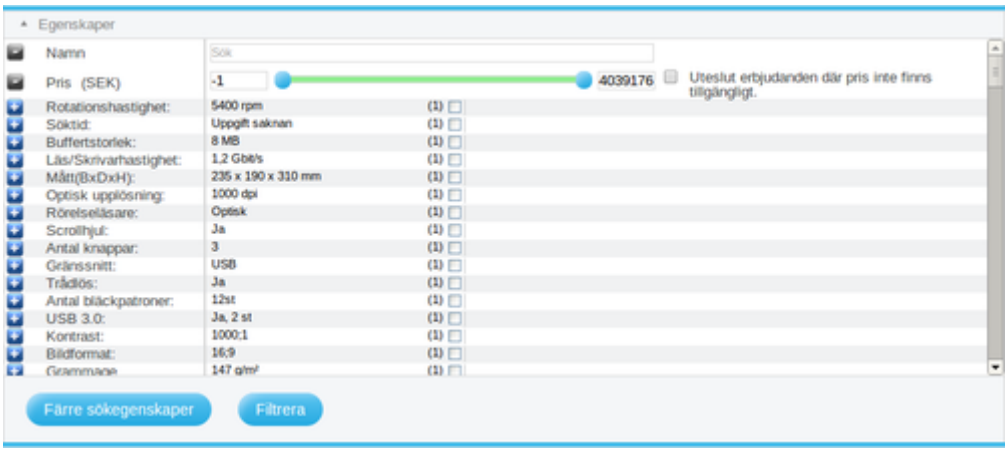


Figure 1: Current property view at Qalixa

Challenge: Empower the user in a complex search space; meta data interactivity By its nature, Information Retrieval involves navigating a large search space. Most of the time, the user is left without a map other than that of intrinsic experience derived from previous interactions with retrieval systems as well as his/her mental model of various searchable concepts. There is little aid for perceivable affordance from the outset. In many enterprise search solutions, or verticals, faceted search is a way to provide a map over the search space. The specific qualities of such map in the Qalixa case will need to be exploited.

For text input alone the visual complexity is relatively small. But for faceted search among an arbitrary amount of properties the GUI can become crowded. There needs to be ways of lifting out the most relevant properties given the information available.

One particular interactive element from the current Qalixa web site can be examined for this particular challenge. When browsing by category, a property view can be called up. The property view allow the user to specify preferences and to filter out unwanted items from the set. Visually, the user is presented with a table with a row for each property and cells for each value. Such presentation is common for such use case. However, for some categories the table become impractical as is grows to hundreds of rows (See fig 1).

Challenge: Performance and latency In order to convey affordance, a tight feedback loop is required. Low latency is thus a high priority in any Information Retrieval application. Qalixa has few special characteristics in this regard. Current solutions are deemed inadequate in this regard. While a throughout performance profiling investigation of the current solution likely could help alleviating the situation it is chosen to be outside the scope of this thesis in favor of

investigating alternatives. It is deemed that a technology switch is required for solving those problems.

Challenge: Information structure and coherence; meta data coherence The problem of structuring data is in large ubiquitous to Information Retrieval but takes a certain shape when addressing faceted search. Certain problems, like finding synonyms and homonyms, are more pronounced. Merging properties can be seen as a superset of the problem of merging tags. Other related problems are automated categorization. Automated categorization can be applied to documents given statistical models where content is analyzed for finding a likelihood of an object belonging to a certain category. For this case, there is the additional problem of categorizing the meta-data itself, like properties.

While this project will only briefly test out a possible solution to this problem, some effort has been placed into looking for potential future work and recommendations.

4 Methods and Tools

Some inspirations that went into the concepts produced are described below. Concepts such as affordance, precision and recall are important. Additionally, some heuristics were used as inspiration as well as for evaluating the prototype. The heuristics draw their motivations mainly from the idea that a good user interface should tax the attention capacities of the user as little as possible for any given task and that any task completion should be made quickly and with few and small steps.

4.1 Usability - Affordance

The concept of affordances was introduced by James J. Gibson and is commonly interpreted as *action possibilities in an environment given an actors capabilities*. It can be divided into a few subcategories, perceived, as affordances that is known by the actor, hidden, as those that are not perceived and false affordances, as those that are perceived but not actual. (1)

4.2 Information retrieval - Precision and Recall

Precision and Recall are central concepts for evaluating information retrieval systems. Precision denotes a measure of the fraction of returned results that are relevant to the query. Accuracy denote the fraction of all relevant items that are contained in the result set. These terms imply a common method for evaluating IR systems, by measuring precision and recall of IR systems they can be evaluated against each other. (2) However, for testing purposes, this method requires a pre-made gold standard. A set of items and queries with relevancy mapping.

4.3 Faceted search: categories, tags, properties and values

Faceted search is the name of an Information Retrieval related HCI methodology where the user is allowed to navigate the search space by selecting properties and values in order to narrow down the search result. Technically any field with a value belonging to a document in the search space

can be used as a *facet*. Examples of facet fields can be “keyword”, “tags”, “length”, “description”, “color” etc.

4.4 Introducing a few usability heuristics

When interacting with an information retrieval or information exploration system, the user is occupied with the task of finding one or many sets of information. It is of interest to the user that this task is made as simple as possible. An interpretation of what simple means in this context follows. But first, we point out a few complementary heuristics.

A graphical interface present a number of elements to the user. An element here constituting a single unit of perceivable information and sometimes one or more perceived or hidden affordances. Each piece of information adds a constant amount of complexity to the overall complexity of the interface. For each actual affordance added, there is also a non negligible amount of complexity added, although this depends more heavily on the users perception. We should distinguish between easily measurable structural complexity and the more elusive perceived complexity perceived and felt by an agent.

Non perceived affordances should not in themselves cause added perceived complexity and mental load. Potentially, perceived and well understood affordances should not cause very much affordances either. Mental load is instead expected to be caused by elements not communicating their affordances well enough, or in other words, the actual affordance can not be unambiguously determined. It is expected that a perceived and well understood affordance will add a small constant mental load per item while a affordance that is not well understood could add a considerably higher load, in worst case causing the user to give up.

Making sure that the maximum amount of hidden affordances are kept low and at the same time allowing such to be easily explored through interaction should help the user to quickly lower the perceived complexity of the application. An affordance should be easily explored if similar actions produce similar results and if the variation of perceived possible actions is low.

For the sake of this project, designing a *simple* interface is interpreted to denote a minimization of above described complexity over the course of a task completion. A task can be more or less well defined. It could be described as ‘finding a nice affordable car’ or finding a car of a particular brand from a particular year of a particular color. This distinction, between a more and a less well defined task, is also the distinction between Information Retrieval, for well defined, and Exploratory Search, for less well defined tasks.

Two relevant heuristics can be defined for making sense of this idea of interface simplicity. *The amount of perceivable elements in a given scene* and *the amount of perceivable elements over the course of a task execution*. In order to minimize the latter, the former needs to be minimized, as well as the amount of scenes that needs to be faced by the user before the task is completed.

Simplicity is not the only requirement for a usability however. Understandability is also needed. Understandability can be seen as the ease of perceiving the affordance of an element given the element’s and related element’s presented information.

Another concern is that the user needs to solve his task quickly. Two things can be seen to consume time given the conceptual model so far, technical *latency* for switching and changing the scene, and time needed to perceive relevant elements in the scene.

4.4.1 Relation of precision and recall to affordance

When looking at affordance in the realm of exploratory search (ES) some related heuristics can be found. It should be desirable to encounter as little complexity during a task completion as possible, as described above. Another way of interpreting this in the realm of ES is that we should maximize precision and recall while minimizing the complexity of the interaction.

Now, there can be many interpretations of this goal. One naive and often used solution is to present a single element of affordance with a wide space of action possibilities, or in concrete terms, a search phrase input form. Varying the amount of tools available in the environment could diminish the amount steps needed for finding a sought article, or it could potentially allow a faster growing recall curve.

4.5 Glossary

Lexicography	study or structure of word relatedness on the basis of semantics and word features
Taxonomy	systematic classification of biological organisms, also classification of information entities in general
Semantic	the meaning of a word
Ontology	a study or structure of meaning of words
Homonym	two words of equivalent spelling but with different meaning
Antonym	word with opposite meaning to another word
Polysemy	word that can have multiple related interpretations

4.6 Literature study

Various articles where studied to acquire inspiration and finding potential inspiration for the Qalixa case. A few where selected for having future potential for the case.

(3) demonstrate methods and analyze a prototype implementation, CREW, for using Wikipedia as a source for creating semantic data and deriving homonyms and synonyms for tags. This could be used for processing untidy data from third party sources and to aid single article publishers to clearly define what they mean. Also, building taxonomies and categorizing new articles could potentially benefit from such technology. A cited article (4), goes further into details on how to obtain *semantic relatedness* from Wikipedia's network of links.

(5) details various methods for semi-automatic clustering; giving the user the ability to influence how clusters are formed through a proposed user interface. This could be useful for giving retailers and users the ability to control how their articles are categorized. (6) suggest another method that focus in particular on leaving the labeling task up to a supervising user.

(7) design and analyze a GUI application, *Stuff I've Seen*, for managing search history in combination with a faceted search. The application allows users to conveniently search through articles previously viewed. The application is evaluated with extensive user tests.

(8) try to attack the problem of information overload by utilizing clustering techniques and corresponding GUI design. A clustering based visualization GUI for navigating a large set of items is presented. The problem of discerning concepts of varying significance is addressed by a *fisheye* concept where more significant words take up more space in the GUI.

(9) present an optimized variant of Lloyd's algorithm using kd-trees. Lloyd's algorithm is a popular algorithm for clustering (10).

4.7 Tools

While there are numerous examples of open source frameworks for information retrieval, and while a exhaustive list is hard to find and probably impractical to maintain, a few frameworks came out as interesting.(11)

A few distinguishing traits can be seen among the alternatives out there. Firstly, there are either open source or closed source alternatives. Closed source alternatives where dismissed at an early stage. The motivation could be that it would narrow down future freedom for the project and Qalixa if put in use, another motivation is that of project scoping and the need to being able to quickly evaluate the solution.

Secondly, many of the search systems out there seems to be either ports or projects building upon Apache Lucene. Examples of the former are Ferret(12) and Lucy (13). Examples of the latter are Solr and ElasticSearch.

Thirdly, frameworks are either built upon an existing database technology, or built as a database system of its own. Lucene and its derivates is an example of the latter, while Sphinx and Riak search could in part be seen as the former (14). Also, Hibernate search could be grouped with the former with a stretch.

4.7.1 Lucene and Solr

All in all, the Lucene family was perceived as the group of engines with the most prevailing community support and at a glance, the largest feature set of the engines looked at. Lucene itself stands out in that it is built for being integrated into a host application(15) rather than being stand-alone out-of-the-box deployable like Solr(16). Lucene requires you to at least build a custom data import layer in a jvm language while Solr allows you to design an import layer meeting many use cases in XML. Similarly, handling of the input query and the process of turning it into a query understandable by the framework is slightly more involved or more manual in the Lucene case. For Lucene, components are combined in code while Solr lets you combine components in XML while being guided by a systematic markup style. Lucene arguably gives you some additional freedom in defining the interface to the search framework, while its derivates adds additional features (17)(18).

Features of Solr include faceted search, geospatial search, XML/JSON/CSV REST-like api, XML configurability. Lucene also cover faceting to some extent (19). Among web services that are currently using Solr are Instagram, The Guardian, SourceForge, eHarmony, Netflix, eBay (Germany), digg and reddit (20). Solr supplies means for running clustering on search results

and documents using the Carrot2 clustering engine (21). It is meant to be highly customizable, allowing plugging in implementations of for example clustering algorithms. Solr has shown to give significant speedup for large scale services, such as Twitter(22).

4.7.2 ElasticSearch

ElasticSearch is another engine in the Lucene family and the closest competitor to Solr. While it is younger and such would have had less time to mature, it seems to have gained a lot of traction. It sports a few features that are by some deemed as game changing. It is built from the ground up with distributed search in mind (23) and has been shown to perform significantly better in some cases where a large index was continuously updated while also performing queries (24). Solr is however deemed faster for indices that seldom change. ElasticSearch is made for easy deployment and it doesn't require defining a schema for basic use cases (25). An additional feature is the Percolator, or reverse search, that allows storing queries and then check which queries are matched by submitting a document (26). Among web services that are currently using ElasticSearch are Mozilla Foundation, SoundCloud, StumbleUpon, Klout, IGN, Sony Computer Entertainment (27) and StackOverflow (28). Developer usability wise ElasticSearch is customizeable by the same JSON REST interface that everything else is passed through. This could be a potential advantage to the rigid file XML file structure of Solr (25).

4.7.3 Sphinx

Another search framework is Sphinx. Some distinguishing features when compared to the Lucene family are tight integration to relational databases, for example MySQL, and a focus on speed, being seemingly faster than Solr for some tasks such as indexing and database queries(29). In fact, one benchmark, albeit potentially outdated, shows Sphinx consistently outperforming Solr in terms of both memory and CPU (30). Sphinx's facilities for faceted search are a bit different. Allegedly allowing a greater set of use cases but with seemingly a bit more manual work before running(31). Another distinction is the source code licence. Solr is Apache licenced while Sphinx is licenced using GPL2 while the maintainers offer commercial licensing as well. Sphinx is used for such services as Craigslist and Groupon and is known to work with more than 25 billion documents and more than 300 million queries per day (32). Sphinx is written in C++ which may bring additional entropy to a Java centric project.

4.7.4 Solr extensions and supporting applications

Some tools that can be used to extend Solr is worth a mention. SolrJ is a java library that serves as a java search interface to Solr that is otherwise talked to using a REST API. Using SolrJ some of the flexibility of Lucene can be recovered (33). Tika is a parser library designed to work with solr, although it is mainly aimed at extracting data from documents (34). Zookeeper is a framework for coordinating distributed processes. It is planned to become an integrated part of Solr for version Solr 4(35). Apache Nutch is a search engine solution that adds a crawler and integrates Tika with Solr (36).

4.7.5 Search framework comparison

For this project, Solr was selected. This is not an obvious choice however. Three frameworks are seen as runner ups - Solr, ElasticSearch and Sphinx. It is appreciated that the choice of one very much depends on the problem. On the basis of scalability, ElasticSearch seems like the best choice at the moment, due to it being built with distributed storage in mind. However, Solr is currently spearheading an initiative to integrate cloud capabilities more tightly (37). Solr sports its clustering engine. Although clustering could be seen as a feature of uncertain current footing in the domain, it may hold future potential. Solr is arguably the platform of choice for experimenting with such techniques, given its Carrot module. Performance tests of Sphinx show promise and the fact that it is used by some large scale e-commerce vendors could be enough to motivate testing it out.

4.7.6 Search frameworks as NoSql stores

Solr and its cousins are structurally very similar to NoSql data stores (38). The distinction is arguably one of marketing strategy (39) although there may be some distinctions in the feature set as well (40). Likewise, some data stores implement many features of search frameworks (14).

4.7.7 Backbone

Backbone is a javascript module and framework for lightweight MVC modeling. It is regarded as a library rather than a framework given that it imposes little restriction on the software architecture(41). It supplies classes (or an approximation for classes) for such things as Collections, Models and Views and facilities for controlling event propagation between them.

4.7.8 Pure Functional and Functional Reactive programming for client side web applications

Some time for shallow testing was given to the Elm language (42), a new programming language that borrows ideas from the pure functional world of Haskell and that focus especially on the idea of Functional Reactive programming (FRP) (43). It compiles to Javascript and is meant to be used for developing graphical web clients. While a more throughout investigation of the pros and cons of using such a language for a complex web application is left outside the scope of this project, a few remarks can be made: The succinctness of implementations of some recurring web application features when using Elm can be demonstrated in the examples on its official web site (44). The Flickr Api example should be the one most relevant to Information retrieval clients(45). However, small implementation tests showed that debug printouts can be very sparse, potentially making continuous implementation challenging.

An alternative to Elm that was also tried to some extent was Fay (46). Fay takes the approach of implenting a subset of Haskell for compilation to Javascript. A second important design choice is to make the code generated as readable as possible, potentially making debugging easier. However, Fay does not give the FRP paradigm any precedence, but there is an example on how FRP can be realized using javascript FRP libraries in Fay (47).

5 Suggested concepts

A few concepts where produced, some of which were made into mock-ups.

5.1 Visual mapping of articles to axes

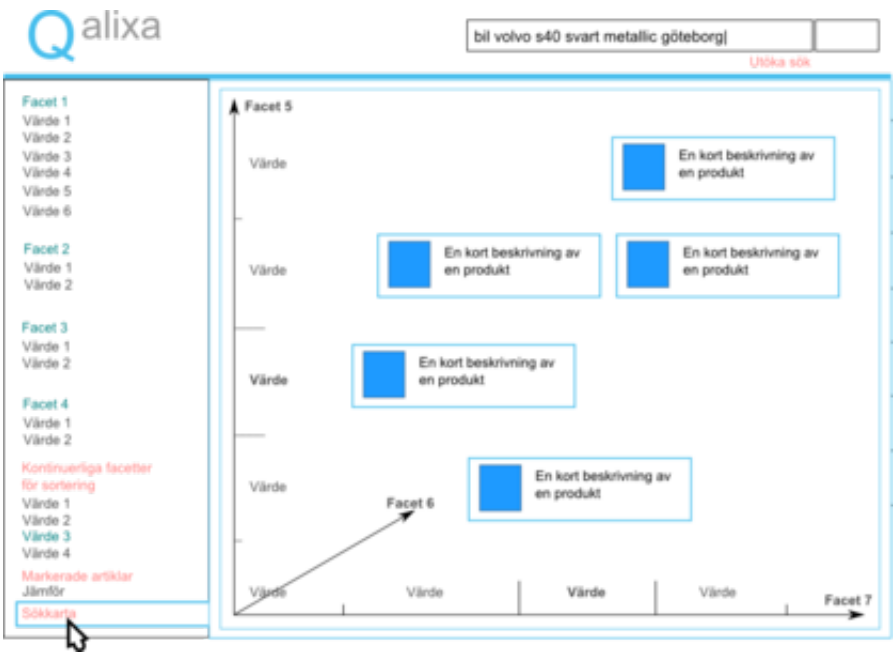


Figure 2: Visual mapping

Some initial ideas explored putting search result items in a map like context, in order to visualize the grouping of items along facets in a continuous way. This could be particularly effective for facets that are of number type (See fig 2). Such feature can allow the user to attain an overview of the distribution of properties over a dimension and also allow faster navigation by supplying an additional empowered level of freedom.

Additionally, through interaction, drag and drop or other, the user could be allowed to control both what facets are bound to each dimension; as well as what value should be brought into view if the interval would not fit on screen. The later could either be controlled by dragging the values along the axis or by dragging from a list of values to the axis (See fig 3). The motivation for such feature could be that it would make it easy for the user to switch what dimensions are to be traversed when selecting out of the result set, and as such, allow the user to navigate a high number of dimensions more quickly.

5.2 Incorporating multiple axes navigation into result list

A variation of axis mapping of facets is to have columns for each result group or facet value in an ordinary result list. Such solution could arguably be more friendly to an adaptive layout while

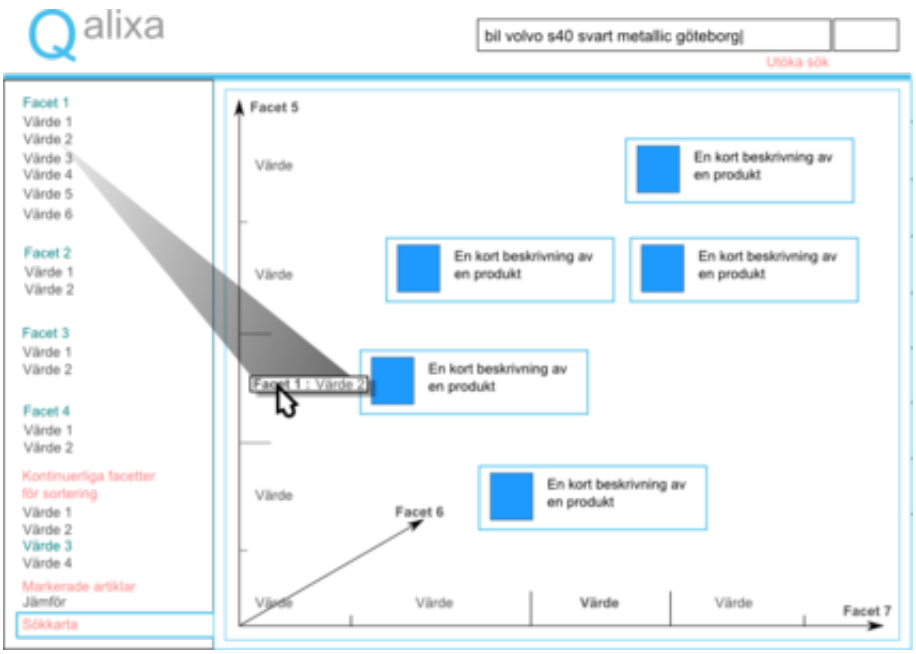


Figure 3: Visual mapping 2

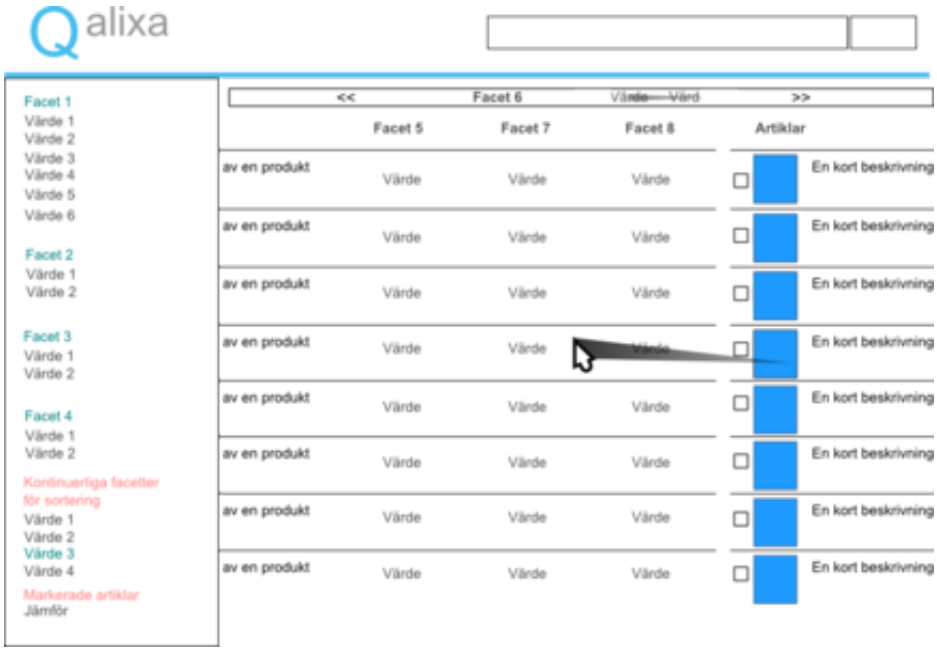


Figure 4: Horizontal drag scroll on facets

somewhat loosing the aspect of communicating the distribution of items along a dimension (See figure 4). It would still have the benefit of supplying an additional degree of freedom that can be used for navigation.

In order to make scrolling to adjacent groups drag scroll could be used. This can expand the interactive region as the user does not need to access periphery scroll controls and doesn't need to know about manipulator key such as shift-scroll. However, drag scroll has so far been avoided for most desktop inclined application; probability for acceptance might be limited.

5.3 Preference collection

Also referred to as filter tray. In order to separate various tasks and as such make the selection routine for large heterogeneous data sets less daunting, selection of properties and selection of their values can be made separate. As suggested by figure 5 properties would reside in a separate element from the so called filter tray. The filter tray would house those properties selected by the user to perform filtering on the result set. Each filter represent a property and supply the necessary controls for selecting a scope of values for that property to select on. Such preference collection can have the potential benefit of both reducing the number of elements shown and to bring the properties useful to the user closer at hand.

If expanding on the concept, there could be ways for the user to build customized collections of preferences. Perhaps as the user return to a category previously visited, the properties and values that was used last time around could be already selected and appearing in the filter tray.

5.4 Creating custom filters by dragging from data elements

As a proposed solution towards easing the process of narrowing down the result set; the user should be able to easily find and select properties and values. Properties may not always be findable in one place, they may live in various spaces of the GUI. The information displayed in the GUI comprise articles, properties or property values. All those elements could potentially be used for rephrasing a query. Properties and values could be made into filters and articles could be seen as a proxy for other keywords, properties or values; either from looking at its own properties, or by looking at its percolated values (See Percolation on article creation).

This gives the idea of allowing the creation of filters from such elements in order to ease exploratory search. For example, values present in the columns of the result list could be made draggable and turned into a facet filter when dropped in the list of filters (See figure 5).

5.5 Dynamic filter lists

Filters, as mentioned above, can be managed in various ways to further lift the capabilities of the user. Rearranging such lists could be made meaningful. One proposition is to make the rearrangement of filter lists to influence the sorting of results. The top most filter, if it represent an interval, should decide the primary sorting feature. Consecutive filters will decide the sorting of items within each group of items that share the same value of the facet represented by the previous filter. For example, if there are two filters, 'price' and 'length', results will be sorted by price first and length second.

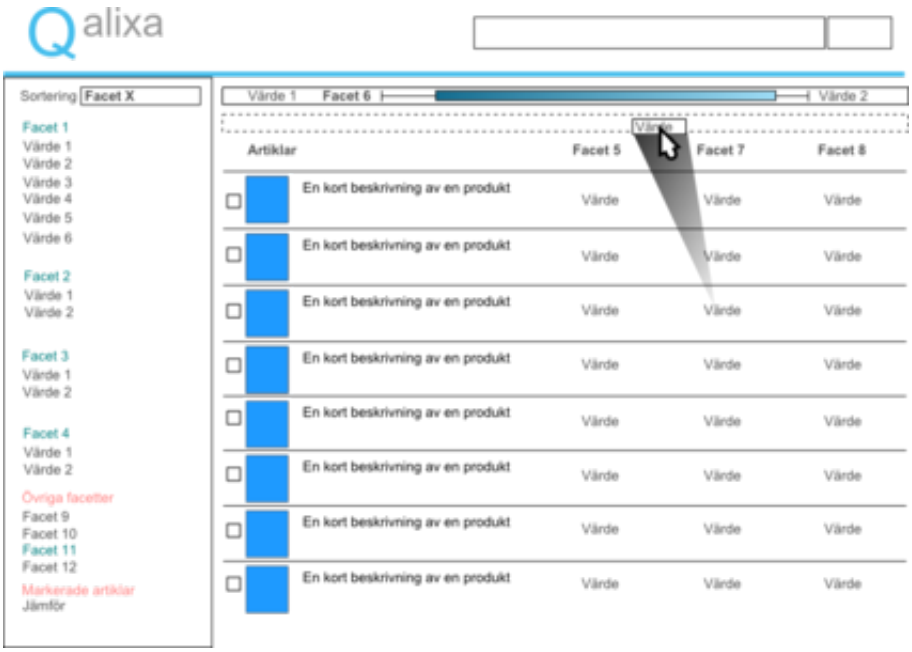


Figure 5: Drag a value to create a filter

A second potential of *dynamic filter lists*, could be that of adapting the selectable range of values based on what ranges are currently available in the result set. If the price range is changed in the price filter, some part of the length range showing in the length filter may no longer be valid. As such, its selectable range should be adjusted. However, this may prove problematic as it can sometimes be desirable to show ranges that are no longer applicable. A solution to that however, could be to just 'grey-out' values that correspond to empty result sets.

5.6 Expand with keywords

Another venture that could be explored is that of adding keywords to a query, as a lightweight way of exploratory search. There should be a fair amount of semantic reasoning behind the suggested words however. Direct synonyms should be searched implicitly and not be part of the suggestions. So suggested words should be related but not equivalent. Additionally, suggestions could be used in order to select among homonyms (see fig 7).

In order to meet the need for semantic relatedness data, above suggested technology for using Wikipedia to that end could be used. While many other applications of the Wikipedia technology likely are possible, it is left out of the scope of this thesis.

5.7 Percolation on article creation

Percolation, an ElasticSearch specific feature, could potentially be used in a scenario where a user is about to create an article for a product, to inform the user how to expose an article for the most relevant search expressions. Percolation works by indexing selected queries so that

Figure 6: Expand search using related keywords

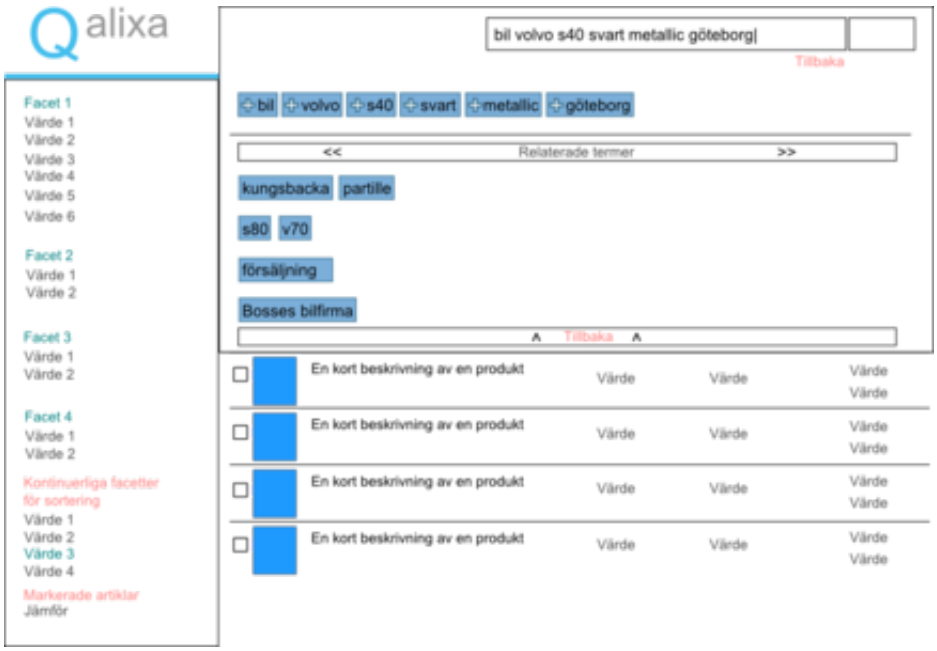


Figure 7: Expand search using related keywords

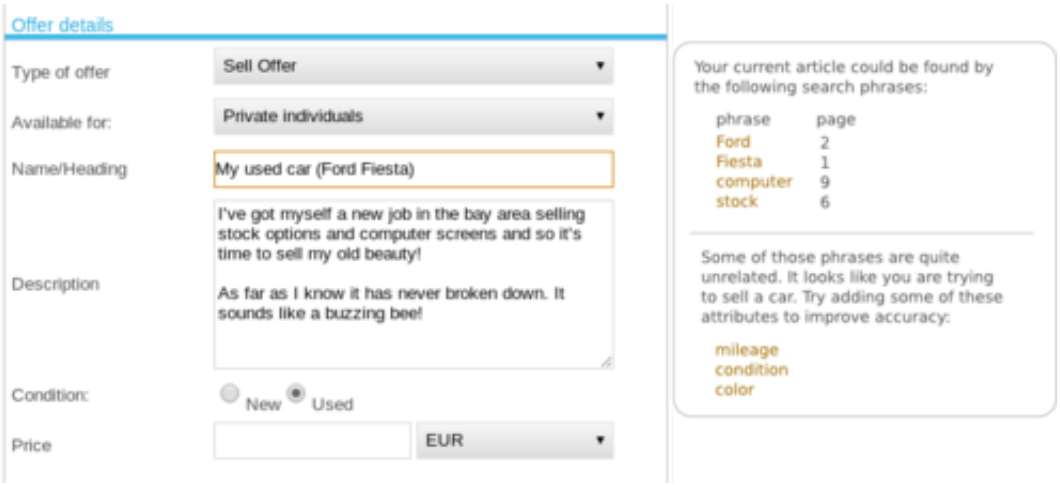


Figure 8: Percolation concept

documents can be tested onto them, seeing if using such query would turn that document into a valid result. This notion could be used to incentivise the user into tagging and otherwise writing the article for maximum precision. Although care should be placed in avoiding incentivising tag abuse instead. By only showing a few queries most relevant to the article this should be avoided. (See fig 8)

Further, percolation could be used for analyzing the impact of entire product feeds coming from retailers. If used as such, it could help incentivise retailers into delivering cleaner feeds and also conveying value brought by the search engine. Also, manual tagging and categorization of multiple products at once could be aided. (48)

Other uses could be thought of as well, but require further feasibility studies. Percolation could potentially bu used for aiding in maintaining and creating tags from queries for example.

5.8 Sunflower navigation

Sunflower navigation expands on the expand with keywords concept and tries to map it onto dimensions; creating a graphical vector representation of each significant keyword in the query phrase and overlaying related, not yet selected, keywords, allowing the user to turn the suggestions into actual search keywords. (See figure 9)

For such concept to work, there needs to be an adjacency heuristic for each pair of tags. Arrows, or vectors can then be put into layout by force based means, to visualize concept closeness. This way, exploring related words can be made quicker as the user is encouraged to first select on subgroups rather than individual words.

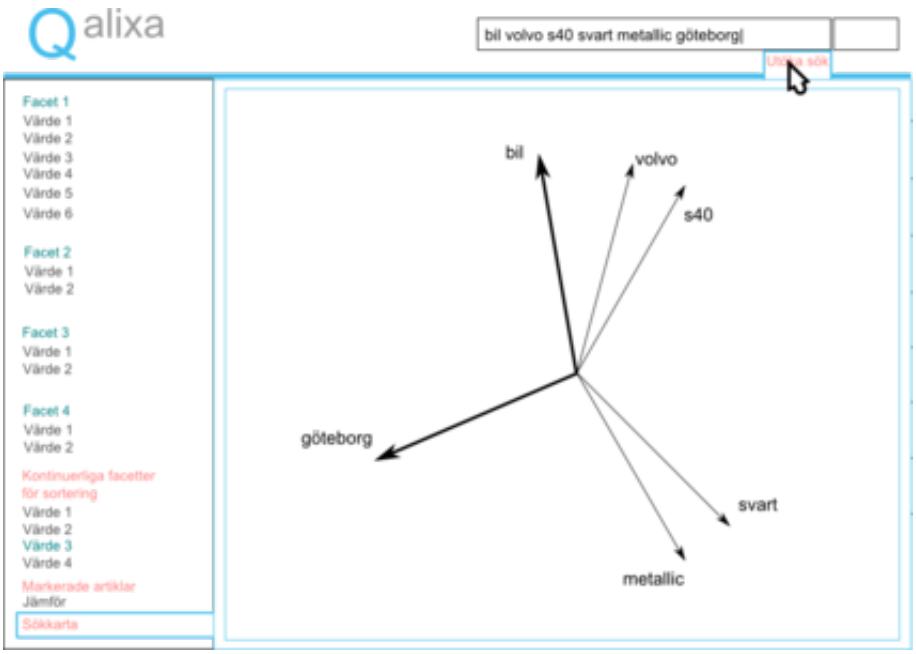


Figure 9: Search map/Sunflower

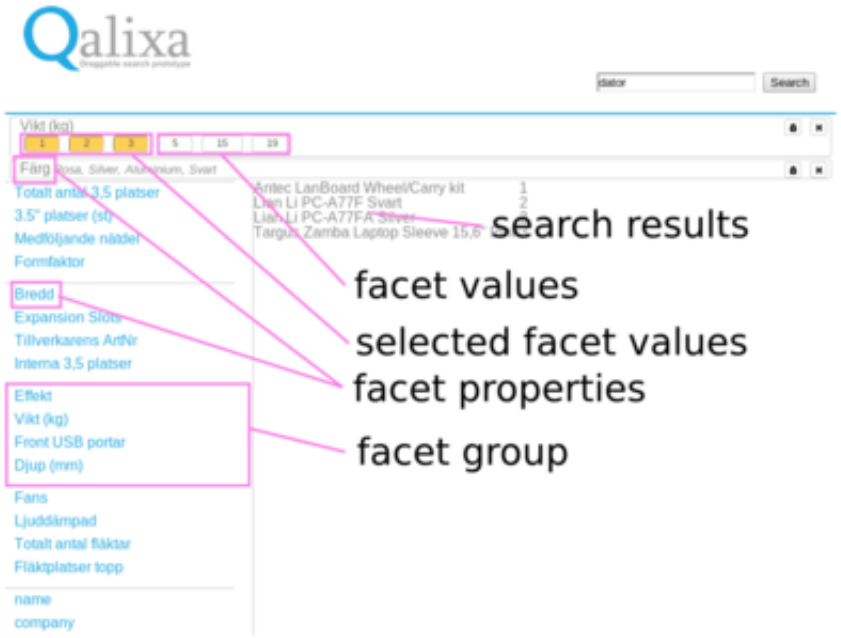


Figure 10: User interface components

6 Prototype

6.1 User interface

The prototype selects from various previously mentioned concepts. Concepts that were focused on were 5.3 and 5.5. 5.4 was tried to some extent but discarded as the overall design did not supply enough useful data elements for such concept to be useful. An additional set of features not outlined above, such as 6.5.5 and 6.5.6, were implemented and are described below.

The client consist of a search form, a side pane populated with groups of attributes applicable for the search phrase currently entered, a list of results and a "filter list". The latter being initially hidden.

The attribute list orders attributes in small groups of some conceptual similarity determined by the clustering algorithm. If the user selects an attribute from the list, it will be added to the filter list. The internal representation of the filter list is traversed when changed to produce a new query once a property value has been selected. Filter components can be rearranged by drag and drop. (See fig. 10)

6.2 Client architecture

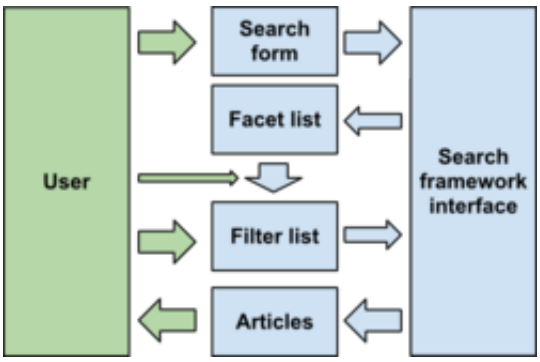


Figure 11: Ideal representation of the application model

An ideal representation of the application model can be seen in fig 11. The search form is the initial interactive component for the user. The search form submit an input phrase to the search framework interface that in turn communicate with the search server. The interface notify both result list and facet list when a response is retrieved. The user can then interact with the facet list to add filters to the filter list. The filter list notifies the search framework when a meaningful change has happened and again the facet and result list is updated. Optionally, filter content is updated as well.

The actual implementation (see fig 12) consist of an ApplicationModel that house all other models except the search interface, SolrInterface. It also handle most of the event bindings and propagations. The application model contain the collection of Attributes, the collection of Filters, the collection of Results, the collection of Groupings and the Query model. The Attributes collection carry the most model logic, collecting all Attribute models. The Attribute model in turn each own a collection of Values that hold Value models. The Grouping and Filter collection

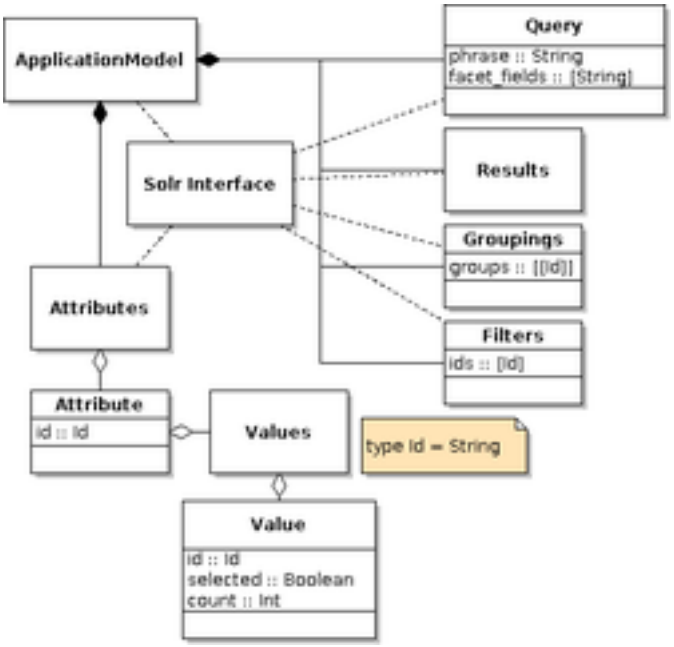


Figure 12: UML representation of the application model

on the other hand only hold references in the form of String ids. At any time when an attribute that form a filter needs to modify its model, it has to be found inside the Attributes collection. This was considered a necessary detail at the time of writing, due to the workings of Backbone, but might be done more elegantly using actual references instead of ids.

There are several view classes used to reflect the state of the models in the GUI. The primary ones are the FacetList, the FilterList and the ResultList. In addition there is FilterComponent, being instantiated as subviews of the FilterList and the SearchForm, representing the form element used for search phrase input.

In addition to the above, there are toolbars. As models and as a view. These are added to allow removing and *locking* filters. The locking of filters denoting locking selectable values from being changed as the result set is changing. (49)

6.3 Algorithmic needs

Looking at challenges that need to be met and the solutions proposed, some algorithmic needs could be discerned. In order to make the data workable and coherent, documents and attributes need to be matched with near equivalents. Synonyms, homonyms and polysemic words need to be recognized somehow. Articles need to be fitted into a suitable category. Articles may be coming from divergent sources, referencing diverse category structures and strictly incompatible taxonomies. Sometimes articles may supply little or no pre existing category assignment and perhaps data to go by.

It should be desirable to automatically extend an existing category structure in some cases, for example when a category is considered to overflow, when there are too many items in a category

Figure 13: Property adjacencies from articles

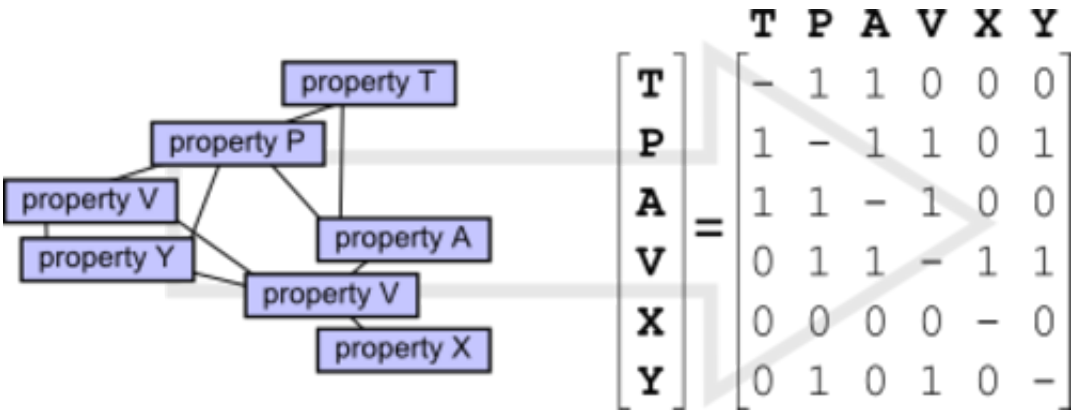


Figure 14: Adjacency matrix

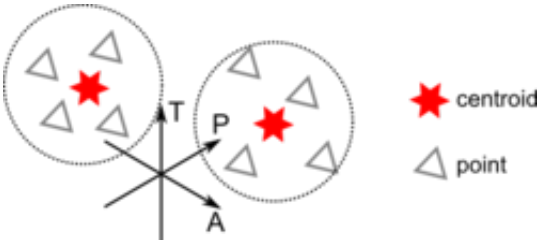


Figure 15: Clustering using centroids showing the three first dimensions/properties

for it being easily navigated. It could perhaps be desirable to show and hide various branches of a complex category tree depending for example on the size of the branches given a search phrase.

Given a large amount of properties given by search results, to be selected from for allowing further refinement, there needs to be ways of give structure in order to lower the complexity faced by the user. Applying clustering techniques could meet this problem to some extent.

6.3.1 Grouping Algorithm

As a partial response for the need of adding some additional structure, a grouping algorithm was implemented. It is is added to the Solr execution as a plugin component to a custom search handler in the Solr configuration. The major part of the algorithm runs as a response to a query, processing the set of articles returned by Solr and producing the relevant output that is attached to the result object returned to the client.

The algorithm begin by generating a adjacency matrix of all the properties of articles of the result set, incrementing the adjacency for every time two properties occur together (Fig 13). The adjacency matrix is then interpreted as an Eucledian space with as many dimensions as there are properties (Fig 14) and passed into an implementation of Lloyd's k-means clustering algorithm, from which a set of groups/clusters are produced (Fig 15). Below is a simplified version of the clustering algorithm in pseudo code:

```
lloyd = function () {
centroids := randomCentroids
until (time_out or goodEnough):
    for each point:
        assignToCluster(point)
    centroids := (for each cluster:
        recompute centroid of cluster)
}

assignToCluster = function (point) {
closestCluster := find cluster closest to point
assign point to closestCluster
}
```

The algorithm is done when the time has run out or when the centroids move sufficiently little per iteration.

6.4 Data importer

A data importer was needed on the server side to correctly and efficiently transfer data from the original MySql database to Solr. While a standardized data import plugin exist for Solr, it was deemed a bad fit for the needs of this project. Import schemes made for the standard importer ran slow and took away the possiblity of multi valued properties. The custom importer showed a very significant speedup. While there may be means of reaching the same speedup without custom code it could not be found.

6.4.1 Modules

The import handler consist of a worker and a front end class. The front end class, FlatTableImportHandler, is called from Solr when there is an url request to the corresponding request handler specified in the Solr configuration. With the url a command is passed. If the command is "full-import" the worker is started in a thread of its own. Any other command, or the lack of a command will return a status report for the worker.

The worker, ImportWorker, execute a MySql query specified in the Solr configuration and read the response line by line. As long as a row has the same unique id, properties specified by the row will aggregate into a document. The document will be submitted to Solr once the id has changed. This is done so that the document in the Solr database will not risk being overwritten.

6.5 Feature roundup

6.5.1 Filter selection and deletion

The user specify property preferences of the sought results by chosing properties from a list. Chosen properties will turn into filters that also display applicable values for the result set, allowing them to be chosen from. Choosing one or more values will cause the document set to be filtered to match the property-value pairs chosen. In non fuzzy mode, it should match at least one property-value combination per filter to not be excluded.

6.5.2 Switchable filter value updates (Padlocks)

In order to give a dynamic view of what properties remain meaningful as preferences are specified, the padlock button allows switching whether the set of values for a particular property should be dynamically updated as the result set changes. The set of values that remain if the filter is *unlocked* corresponds to those property values that can be found in the result set.

6.5.3 Sliders for enumerable facets (deselecting intermediate values?)

Some properties are recognized as *enumerable*, or of number type. It should in most cases be interesting for the user to select a span rather than individual values for such cases, avoiding the extra work of selecting a large number of similar values. Therefore, a slider is used for controlling such value selection.

6.5.4 Switchable sorting and sorting customization by drag and drop

Without adding additional elements to the scene, a fine grained control for the sorting order of results is made possible by looking at the ordering of filters. For an attentive user, it can be seen that the sorting order is made by each column in order. The rationale for this is that the prioritization of the filters, or *preferences* of the user, can be signified by the visual order. The order of the columns in the result list is determined by the order of filters as well. The filter order can be rearranged by drag and drop.

6.5.5 Dynamic column creation

What properties are to be displayed in the result list is determined in part by a predefined list of default properties and in part by the properties corresponding to the filters in the filter list. This allows a dynamic display of articles where focus is put on those properties that are likely of most interest to the user.

6.5.6 Fuzzy facets

The user has an option of enabling *facet fuzziness*. This causes the filters to become promoting/demoting of matching/non-matching articles rather than excluding filtered articles entirely. This feature is realized with Solr using query boosting. For example, the following generated query will only retrieve articles that match the preferences exactly:

```
car AND (color:blue AND price:[100 TO 10000] AND model:Toyota)
```

However, adding wildcard queries with a lower prioritization/boost will allow previously unmatched articles to appear towards the end of the result list:

```
(car AND (color:blue^1000 OR color:[* TO *]^0.1)
     AND (price:[* TO 10000]^0.9 OR price:[* TO *]^0.1)
     AND (model:Toyota^1000 OR model:[* TO *]^0.1))
```

There is a reservation on this solution; articles that does not define the property will still not be included. Depending on the reliability of the data this may or may not be desirable. An addition that should include even such articles with an even lower prioritization would be as below. This possiblity is not demonstrated in the prototype however.

```
(car AND (color:blue^1000 OR color:[* TO *]^10)
     AND (price:[* TO 10000]^0.9 OR price:[* TO *]^10)
     AND (model:Toyota^1000 OR model:[* TO *]^10))
 OR car^1
```

7 Analysis

7.0.7 Intended use case

The intended use of the prototype is to first enter a generic search phrase in the search form and hit 'Search'. Secondly, zero or more properties are selected from the left according to one's preferences. Thirdly, selected properties, now turned into filters in the filter tray above the result list, can now be used for narrowing down the results. Selecting multiple values from the same property will cause a non-exclusive multi-value selection on that property. Or in other words an 'OR' selection will be made with all selected values in one particular property. If multiple filters are used, an exclusive selection will happen, combining the filters with 'AND' operation.

The intention is that both retrieval of specific known articles, using known properties of such articles, and exploration of the data set within a certain domain, should be made easier. Allowing

direct feedback when alternating both properties and property values should allow exploration while supplying relevant properties should allow the user to quickly zoom in on an intended article. Additionally, adding and subtracting filter selections should allow the user to identify corner cases, highlighting what articles the user is opting out.

7.0.8 Observed issues

Some issues can be observed when looking at the prototype. Issues that are considerable in scope but that if solved, should significantly elevate the potential of the approach suggested by the prototype. Some of these challenges are arguably hard to avoid for any attempt at dealing with e-commerce from the perspective chosen by Qalixa and similar ventures, and should emancipate considerable value if solved comprehensively. Many of these issues are expected to be especially relevant when having a great number of dynamic properties coming from varying sources.

The most apparent problem in the prototype is the high amount of property fields that are available for being selected from by the user. Many of the fields are nonsensical when taken out of context of their parent articles or seem to be irrelevant for most uses. Many properties and values are duplicates or are of very similar meaning. Sometimes property names can map to values of varying type or dimension. Some values may be of number type but this is not recognized and as such, the property is not given a slider when used as a filter. Some properties have superfluous characters in their name. Some values are placeholders for non-values, some should be split into multiple values.

Many properties have only a single selectable value. This is arguably a part of a larger problem. At the point of retrieving properties, the amount of applicable values for each property given the current search space is not known (only the number of occurrences of the property fields throughout the result set is known). Retrieving applicable values for a large result set for all properties is supposedly very computationally expensive, however possible, using the Solr REST API. With the current prototype, only the values of the chosen filters are retrieved.

More subtle are the issues that arise once you start the interaction. One possible issue is that of filtering out false negatives. If a certain attribute is not defined for an item it can not really be known if it really should be filtered out or not. The default behavior of Solr is to filter it out, which may be undesirable. Similarly, a lacking in synonym understanding among values should cause problems of false negatives as well.

Dynamic filter options and *Padlocks* One issue appear as a result of updating selectable filter content dynamically. As the result set is diminished, the set of field-value pairs normally diminish as well. If this is not reflected in the possible preferences of the filters, there is a risk of allowing the user to select a set of preferences that results in an empty result list. In some cases this risk may even be higher than selecting preferences resulting in a non-empty result set.

Always allowing dynamic updates of filters is problematic since the filter mix the trait of excluding existing or including additional articles. When first putting a filter into use, the filter acts to exclude items from the result set. When selecting additional values, the result set grows. Similar growth and shrink pattern is reflected in the set of applicable values and properties. Additionally, whilst using fuzzy facets or working with a large set, the issue is a bit more subtle. Individual articles, properties and values may be excluded by way of becoming less prominent as the set grows. As such, filters or values will become treated as being to insignificant for inclusion, causing it to disappear mysteriously.

The Padlock feature exist as to partially adress this problem. To help the user to avoid these cases the padlock button allows the user to manually switch whether applicable filter values are dynamically updated. This is far from a perfect solution as it requires the user to figure out when to apply the dynamic him/herself.

Summarized, the issues in list form:

- Missing context for understand property meaning
- Irrelevant properties
- Synonyms not recognized
- Unit or qualitative/quantitave homonyms not recognized
- Basic sanitization and interpretation of names and values are lacking upstream
  - Special characters in property names
  - Multi values interpreted as single values
  - Non-value placeholders interpreted as values
- Insufficient structuring of large amount of properties
- Over-eager filtering, false negatives
  - Synonyms
  - No defined value for selected property
- Invalid or missing options among filters after query

7.0.9 Technical issues

Some issues are purely technical. Some property names cause problems due to Solr not being able to cope with special characters. This could be blamed on insufficient sanitization but there may be cases where special characters are desired in the displayed name. Additionally, many non-english languages will require support for Unicode. For these purposes Solr have some support, using the ASCIIFoldingFilterFactory (50) or UnicodeCollation (51). These are not applied in the prototype however.

Another issue is the need to associate and infer additional information to the property name, such as property type. The variation of property types may for example be used for varying visualization and interaction methods, so that properties with number values may use sliders. Associating additional data to field names may require creating additional entries to the database. Such entries could also associate a display name with special characters with a look up name, without such characters, if needed.

7.0.10 Comparison to current property view of Qalixa.com

Previously, the current element for selecting among properties and property values were described briefly (See fig 1). Analyzing the current solution while taking inspiration from the suggested usability heuristics we can make the following observation. For the task of allowing the user to select preferences given applicable properties the current solution present a number of visual elements roughly equivalent to the number of properties times the number of values on average. The proposed solution present a number of visual elements roughly equivalent to the number

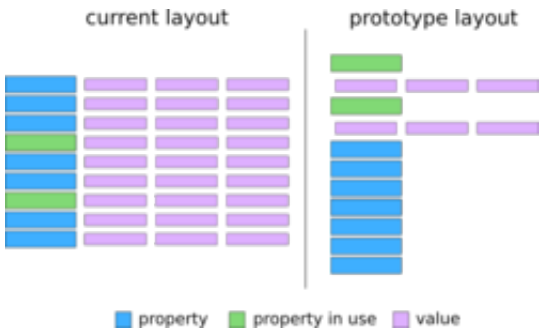


Figure 16: Diagram comparison of visual complexity

of properties plus the number of selected properties plus the number of values of the property currently being in focus. This amount is strictly smaller in general and is expected to be significantly smaller for most cases (See fig 16).

Furthermore, the ratio of visual elements shown clearly to the user, high up on the page, that are of direct interest versus such elements that are of secondary interest, should also have improved significantly. *Filters* corresponding to the users preferences are maintained in plain site and does not remain among the properties in the property list.

8 Discussion

The prototype tested a few of the ideas presented to some degree. Creation of custom filters where tried to some extent, but discarded due to adding little purpose to the prototype in its current form. The most prominent concept in the prototype is that of the filter tray. Dynamic filter lists were used as well although it was hard to make conclusions about its usefulness.

Apart from the suggestions, the project served to investigate the potential of Solr and similar technologies for Qalixa and it brought a variety of new lessons on web technology, such Javascript frameworks, databases and search engines, to its author.

Given that relevant properties can be found in the property list and given that they house sensible values, some observations can be made about the benefits of the prototype and its approach.

Once properties have been selected, distance in between useful components is small. This is beneficial for exploring small variations on the same properties. Having sliders for number properties allows for fast exploration. Having low latency for queries when switching properties on and off allows for a fast feedback loop on what selections fall off as the scope is narrowed.

Also it can be seen that the set of results are quickly narrowed down as filter scope is narrowed down. The problem of drop off due to false negatives will still be a problem even with fuzzy faceted search though. There will be likely be a bias towards articles that has well defined properties. Although with fuzzy faceted search, recall should become sufficient. Precision due to variations in how article properties are defined are to be expected. On the other hand it is arguably so that an article is more likely to be incorrectly excluded/demoted than included/promoted when filtering on a property. Overlooking the act of setting a property seem more likely than setting the property wrongly.

It is observed that being able to quickly alter between adjusting various property value scopes is useful. This is partially accomplished by the prototype. Although the later prototype only allows having one active, as in adjustable, filter at a time. Earlier snapshots allowed filters to be expanded and collapsed at will. This makes the scene slightly less complex but also hinders switching between filters quickly.

Having interesting filters clearly separated from the overall property list also seem to make user preferences manageable. Although here is an ambiguity regarding the persistence of filters. What should happen when the filters are no longer relevant for the search phrase? The current behavior of the prototype is to remove such filters. This can sometimes be problematic. A better choice is perhaps to render those filters inactive and mark them as such. Allowing the user to remove them at will.

There was an early attempt at allowing the user to select filters by drag and drop, as suggested by the suggested concepts above. This was deemed to supply little value and some distraction for the user, and was discarded in favor of simply clicking elements for turning them into filters. A more complex web of possible interaction potentials between various components, where elements can move between various parts to a greater extent and in a meaningful way, could make drag and drop useful again.

The order of the result list is in the prototype determined by the order of used filters in the filter list. While this may be a useful feature in some edge cases, it is deemed that this is not a practical default behavior that communicates badly to a user. Having explicit controls to enable this behavior may be useful, although it is likely that the de facto default behavior of most result lists out in the wild, to sort if the column header is clicked on, is more intuitive due to its strong history. However, the method suggested here have the potential advantage of sorting on multiple criteria. This could be useful but likely only in periphery use cases.

8.1 Suggestions, for academia, industry and for Qalixa

To establish structure and coherency of the meta data, a few things are suggested for further investigation and for solving related problems in broad terms.

8.1.1 Pruning and sorting properties using statistics and traits

Two methods for grouping properties are recognized. Properties can be grouped by categories and grouped by traits. *Category* here denoting and assuming a preexisting grouping of articles in a category tree. Such structure can be carried over to the properties so that each category is tied to the categories that occur in the articles that are sorted into it. If a category has been selected by the user, properties can be shown based on whether they belong to that category as properties that exist outside the category are likely irrelevant. Further, properties that are part of a subcategory or a smaller subset of articles in the current category may not be very relevant either as a filter that would use such category would only be applicable on a small part of the articles. A generalization of this idea would be that of finding applicability of facet properties in a given selection, regardless of having a stated category. Bear in mind here that it might not only be properties that can be found in the current selection that might be of interest. Similarly to how non-selected values may be of interest, non-used properties may be of interest too, once the user looks to expand his/her preferences rather than diminishing them.

We could introduce a notion called *facet traits*. A facet property may have more or less values, and values may be of certain types. A property convey a number of applicable values for a given selection, the values that exist in the selection, but other values that may be of interest may exist throughout the database. Facet traits may be made to connote the characteristics of the its values, throughout the database on one hand, and given a selected context, on the other. Such traits can be used to further determine how a property is to be presented. The *facet trait* idea can also be extended to connote the above described property characteristics. To how large extent of the current selection does the property exist? To how large extent of the entire database and in what categories does the property exist? So such traits are in part consistent regardless of context and in part context dependent.

Having such trait information available could help in building the presentation and allow qualified decisions on what to present and how to build user interface elements. Examples of using such traits could be:

- Turning single valued facets into keywords
- Hiding facets that exist for a small subset
- Show facets that does not qualify for the current selection but for a relevant superset, for example a category

8.1.2 Controlled folksonomies

Having folksonomies, or wiki-like tag systems can be a useful way to establish structure in a large data set where items are interacted with by users on a regular basis. But to find the best way to harness it can be a challenge. (52) explain different algorithmic methods for extracting good tag sets from folksonomies. Taking synonyms and similar problems into account.

8.1.3 Client side facet search

in order to elevate the potential of facet search, and to allow fluid filtering for the user even if the server load is considerable, using client side facet search could be desirable. There are implementations for this available that show great promise. While client side search could not realistically cover the breadth of a server based solution, it can serve as a complement that works on narrowing down the set of results given by the server. The server could be allowed extended latency and fill up with new results only when the client side result set is draining out. (53) (54)

8.1.4 Grouping facets by outsourced semantics

One promising approach to enhance structure is to utilize outside sources of semantic structure, so called semantic web technologies. For example, DbPedia (55) is an initiative for extracting semantic data from Wikipedia. A derived application, DbPedia Spotlight, can infer context to words in a text. Inspiration can also be taken from the way a prototype, Sztakipedia (56), helps the user by suggesting meta data to add to an article while being worked on. Such concept could be used in conjunction with the percolation concept suggested earlier.

8.1.5 Annotating rather than excluding invalid filters and values

The prototype combine different approaches in order to deal with invalid filters and values. Filters are removed when no longer needed. Values can either update dynamically or remain untouched. Such approaches are arguably confusing. A better solution, until a comprehensive solution is found, would be to mark options depending on the effect using them would have. Unused but selected filters that represent fields no longer found in the result set could be marked red for invalidity. That way the user can easily backtrack if that property was of interest. Values of an active filter that represent articles that are not in the result set could be marked blue to communicate a potential addition.

8.1.6 Multi touch interaction for facet navigation

A suggestion made by (57) is to use multi touch devices for allowing additional levels of freedom when exploring facets. In the current prototype, only a single preference can be adjusted at a time. The concepts described above suggest allowing moving in multiple dimensions for allowing the user to explore multiple dimensions at once. By allowing adjusting multiple preferences at once, preferably with sliders, utilizing multi touch capability, another variation is possible. The method would however be limited to such capable devices.

9 Appendix

9.1 Source code

The source code for the prototype can be found at (58). It can be deployed and used by following the contained README. The instructions as well as known issues will be repeated below for completeness, but may not be up to date.

9.1.1 Summary of repository contents

All these components are designed to work alone reasonably well. They should depend on each other only run time wise. Data importer depends on Solr. Grouping component depends on the data importer. Client depends on grouping component (although search should work without grouping, possibly you need to disable grouping in Solr config).

Flat table import handler (server/customComponents/dataImport) For importing data from MySql database to Solr

Field grouping component (server/customComponents/fieldGroupingComponent) For clustering of attributes when submitting a search query. Has two grouping algorithms that are switchable by a constant in the code (FieldGroupingComponent.groupingScheme).

Master thesis gui/web client (client) Web front end that talks with solr. Using backbone, underscore, jquery and some jquery-ui.

Documentation Source material for this thesis.

9.1.2 Known issues

As far as known, all of the known buggy behaviors can be escaped by reloading the page.

- The gui does not update if the result of the query was null
- Clustering algorithm is throwing exceptions for some (small or empty?) data sets, causing failure to update
- Selecting some attribute values containing whitespaces or unconventional symbols may cause malformed query to solr, failure to update or wrong results
- Submitting an empty phrase (in search field) will return a result
- The maximum number of imports processed by the data importer is currently hard-set by an if statement in its main while loop. (Intentional for debug purposes)
- The multi valued-ness of properties currently needs to be set in two places in the solr configuration. In the configuration of the import handler and in the schema. It would be better if it could be set only in the schema.
- For some queries Solr responds with a null object. However, at other times 'no documents found' results in a result object with no
- The current schema produce a extremely large database. Consider tweaking the schema and removing long descriptions from index. It is currently not possible to save the entire database to disk.

9.1.3 Set up

9.1.4 Assumptions

- apache 2 running on machine
- mysql 5.6>= running on machine with database adherent to config in server/solrconfig

9.1.5 How to set up server

- fetch submodules
- run getSolr in server folder
- run runServer to start server
- fill up the database by making data import query to solr: localhost:8983/solr/flattable-dataimport?command=full-import
- Check its progress by visiting: localhost:8983/solr/flattable-dataimport
- If the data is begining to take up too much space, you can abort and commit with localhost:8983/solr/flattable-dataimport?command=stop

How to set up client

- Configure apache to point at the client folder
- Go to localhost in the browser
- Use the gui!

Build data importer and grouping module

- You *should* not need to do this to run. There are pre built jar-binaries in server/solrconfig/lib.
- If you want to update the jars. Run the ant build scripts in each eclipse project, default target.

9.2 Additional papers

Here are a few sources for relevant papers that were discovered to late for being utilized in the paper; they should be evaluated in future projects:

- Symposium on Human-Computer Interaction and Information Retrieval (59)

Papers that were not included but may be of interest are: - (60) propose a system for mapping between taxonomies of products in e-commerce applications. - (61) deals with interoperability of databases by correlating schemas using similarity measures. - (62) describe optimizations for spectral clustering, a method for allowing dimensionality reduction that may be relevant for cases where the input set consist of a large adjacency list.

10 Citations

1. WIKIPEDIA. Affordance. In: . 2013. [accessed 25-February-2013] url

2. WIKIPEDIA. Precision and recall. In: . 2013. [accessed 25-February-2013] url

3. LIU, C., OOI, B. C., TUNG, A. K. H. and ZHANG, D. Crew: cross-modal resource searching by exploiting wikipedia. In: *Proceedings of the international conference on Multimedia*. S.l.: ACM, 2010. pp. 1669–1672.

4. WITTEN, Ian H. and MILNE, David. An effective, low-cost measure of semantic relatedness obtained from Wikipedia links. In: *Proceeding of AAAI Workshop on Wikipedia and Artificial Intelligence: an Evolving Synergy, AAAI Press, Chicago, USA*. S.l.: s.n., 2008. pp. 25–30.

5. HU, Y., MILIOS, E. E., BLUSTEIN, J. and LIU, S. Personalized document clustering with dual supervision. In: *Proceedings of the 2012 ACM symposium on Document engineering*. S.l.: ACM, 2012. pp. 161–170.

6. CHEN, K. and LIU, L. Clustermap: Labeling clusters in large datasets via visualization. In: *Proceedings of the thirteenth ACM international conference on Information and knowledge management*. S.l.: ACM, 2004. pp. 285–293.

7. DUMAIS, S., CUTRELL, E., CADIZ, J. J., JANCKE, G., SARIN, R. and ROBBINS, D. C. Stuff I've seen: a system for personal information retrieval and re-use. In: *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*. S.l.: ACM, 2003. pp. 72–79.

8. TURETKEN, O. and SHARDA, R. Development of a fisheye-based information search processing aid (FISPA) for managing information overload in the web environment. In: *Decision Support Systems*. 2004, Vol. 37, no. 3, pp. 415–434.

9. KANUNGO, Tapas, MOUNT, David M., NETANYAHU, Nathan S., PIATKO, Christine D., SILVERMAN, Ruth and WU, Angela Y. An efficient k-means clustering algorithm: Analysis and implementation. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* [online]. 2002, Vol. 24, no. 7, pp. 881–892. Available from: http://www.cs.rit.edu/~rpj/courses/bic2/lectures/kmeansanalysis.pdf.

10. WIKIPEDIA. In: [online]. Available from: http://en.wikipedia.org/wiki/Lloyd's_algorithm.

11. WIKIAUTHORS. In: [online]. Available from: http://en.wikipedia.org/wiki/List_of_enterprise_search_vendors.

12. FERRETMAINTAINERS. In: [online]. Available from: https://github.com/xing/ferret readme. Online; accessed 2013-02-28

13. LUCY, Apache. In: [online]. Available from: http://lucy.apache.org/. Online; accessed 2013-02-28

14. KOVÁCS, Kristóf. Cassandra vs MongoDB vs CouchDB vs Redis vs Riak vs HBase vs Couchbase vs Neo4j vs Hypertable vs ElasticSearch vs Accumulo vs VoltDB vs Scalaris comparison. In: [online]. Available from: http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis. Online; accessed 2013-02-28

15. LUCENETUTORIAL.COM. Lucene in 5 minutes. In: [online]. Available from: http://www.lucenetutorial.com/lucene-in-5-minutes.html. Online; accessed 2013-02-28

16. SOLR. In: [online]. Available from: http://lucene.apache.org/solr/api-3_6_2/doc-files/tutorial.html. Online; accessed 2013-02-28

17. Solr Features. In: [online]. Available from: http://lucene.apache.org/solr/features.html.

18. TAN, Kelvin. Apache Solr vs ElasticSearch. In: [online]. Available from: http://solr-vs-elasticsearch.com/. Online; accessed 2013-02-28

19. Apache Lucene Faceted Search User's Guide. In: [online]. Available from: http://lucene.apache.org/core/3_6_1/api/contrib-facet/org/apache/lucene/facet/doc-files/userguide.html. Online; accessed 2013-02-28

20. APACHE-WIKI-AUTHORS. PublicServers. In: [online]. Available from: http://wiki.apache.org/solr/PublicServers?action=recall&rev=380. Online; accessed 2013-02-28

21. SOLR-WIKI-USERS. Clustering Component. In: [online]. Available from: http://wiki.apache.org/solr/ClusteringComponent?action=recall&rev=61. Online; accessed 2013-02-28

22. TWITTER. Twitter Search is Now 3x Faster. In: [online]. Available from: http://engineering.twitter.com/2011/04/twitter-search-is-now-3x-faster_1656.html. Online; accessed 2013-02-28

23. BANON, Shay. In: [online]. Available from: http://stackoverflow.com/revisions/2288211/3. Online; accessed 2013-02-28

24. SONNEK, Ryan. Realtime Search: Solr vs Elasticsearch. In: [online]. Available from: http://blog.socialcast.com/realtime-search-solr-vs-elasticsearch/. Online; accessed 2013-02-28

25. ELASTIC-SEARCH-MAINTAINERS. Schema Free & Document Oriented. In: [online]. Available from: http://www.elasticsearch.org/. Online; accessed 2013-02-28

26. BANON, Shay. Percolator. In: [online]. Available from: http://www.elasticsearch.org/blog/2011/02/08/percolator.html. Online; accessed 2013-02-28

27. ELASTICSEARCH. ElasticSearch users. In: [online]. Available from: http://www.elasticsearch.org/users/. Online; accessed 2013-02-28

28. CRAVER, Nick. A new search engine for Stack Exchange. In: [online]. Available from: http://meta.stackoverflow.com/revisions/160100/15. Online; accessed 2013-02-28

29. UGS. Choosing a stand-alone full-text search server: Sphinx or SOLR? In: [online]. Available from: http://stackoverflow.com/a/6166581/439034. Online; accessed 2013-02-28

30. WEAVER, Evan. rails search benchmarks. In: [online]. Available from: http://blog.evanweaver.com/2008/03/17/rails-search-benchmarks/. Online; accessed 2013-02-28

31. SPHINX. In: [online]. Available from: http://sphinxsearch.com/docs/1.10/multi-queries.html. Online; accessed 2013-02-28

32. Sphinx overview. In: [online]. Available from: http://sphinxsearch.com/about/sphinx/. Online; accessed 2013-02-28

33. CASSANDRA TARGETT, Elna Tymes and ET AL. Using SolrJ. In: [online]. Available from: http://lucidworks.lucidimagination.com/pages/viewpage.action?pageId=14647724. Online; accessed 2013-02-28

34. ERICKSON, Erick. Indexing with SolrJ. In: [online]. Available from: http://java.dzone.com/articles/indexing-solrj. Online; accessed 2013-02-28

35. SOLR-WIKI-USERS. Solr 4.0. In: [online]. Available from: http://wiki.apache.org/solr/Solr4.0?action=recall&rev=8. Online; accessed 2013-02-28

36. About Apache Nutch. In: [online]. Available from: http://nutch.apache.org/about.html. Online; accessed 2013-02-28

37. SOLR-WIKI-AUTHORS. SolrCloud. In: [online]. Available from: http://wiki.apache.org/solr/SolrCloud?action=recall&rev=93. Online; accessed 2013-02-28

38. INGERSOLL, Grant. NoSQL, Lucene and Solr. In: [online]. Available from: http://searchhub.org/dev/2010/04/30/nosql-lucene-and-solr/. Online; accessed 2013-02-28

39. KARWIN, Bill. Why are document stores like Lucene / Solr not included in NoSQL conversations? In: [online]. Available from: http://stackoverflow.com/a/3339826/439034. Online; accessed 2013-02-28

40. KULLMANN, David. Elasticsearch and NoSql database. In: [online]. Available from: http://stackoverflow.com/revisions/8059103/2. Online; accessed 2013-02-28

41. SANDERSON, Steven. Rich JavaScript Applications – the Seven Frameworks. In: [online]. Available from: http://blog.stevensanderson.com/2012/08/01/rich-javascript-applications-the-seven-frameworks-throne
Online; accessed 2013-02-28

42. CZAPLICKI, Evan. The Elm Programming Language. In: [online]. Available from: http://elm-lang.org/. Online; accessed 2013-02-28

43. HUDAK, Paul, COURTNEY, Antony, NILSSON, Henrik and PETERSON, John. Arrows, robots, and functional reactive programming. In: *Advanced Functional Programming* [online]. 2003, pp. 1949–1949. Available from: http://www.staff.science.uu.nl/~jeuri101/afp/afp4/hudak.pdf.

44. CZAPLICKI, Evan. Learn by Example. In: [online]. Available from: http://elm-lang.org/Examples.elm. Online; accessed 2013-02-28

45. CZAPLICKI, Evan. Flickr. In: [online]. Available from: http://elm-lang.org/edit/examples/Intermediate/Flickr.elm. Online; accessed 2013-02-28

46. DONE, Chris and ET AL. Fay Home. In: [online]. Available from: fay-lang.org. Online; accessed 2013-02-28

47. TRANGEZ, Nicolas. Bacon-n-Fay. In: [online]. Available from: https://github.com/NicolasT/Bacon-n-Fay. Online; accessed 2013-02-28

48. SUCHAL, Ján. Elasticsearch - advanced features in practice. In: [online]. Available from: http://www.slideshare.net/jsuchal/elasticsearch-advanced-features-in-practice. Online; accessed 2013-02-28

49. FREDELIUS, Per. Project GUI source. In: [online]. Available from: https://bitbucket.org/worldsayshi/master_thesis-gui/src. Online; accessed 2013-02-28

50. SOLR-WIKI-USERS. solr.ASCIIFoldingFilterFactory. In: [online]. Available from: https://wiki.apache.org/solr/AnalyzersTokenizersTokenFilters?action=recall&rev=141 solr.ASCIIFoldingFilterFactory. Online; accessed 2013-02-28

51. SOLR-WIKI-USERS. In: [online]. Available from: http://wiki.apache.org/solr/UnicodeCollation?action=recall&rev=5. Online; accessed 2013-02-28

52. DATTOLO, Antonina, EYNARD, Davide and MAZZOLA, Luca. An integrated approach to discover tag semantics. In: *Proceedings of the 2011 ACM Symposium on Applied Computing*. S.l.: ACM, 2011. pp. 814–820.

53. M, Vinay. In: [online]. Available from: https://github.com/rmdort/backbone-faceted-search. Online; accessed 2013-02-28

54. M, Vinay. In: [online]. Available from: http://eikes.github.com/facetedsearch/. Online; accessed 2013-02-28

55. DBPEDIA. In: [online]. Available from: http://wiki.dbpedia.org. Online; accessed 2013-02-28

56. HÉDER, Mihály. In: [online]. Available from: http://pedia.sztaki.hu. Online; accessed 2013-02-28

57. FJELD, Morten. 2013. S.l.: personal communication.

58. FREDELIUS, Per. In: [online]. Available from: https://bitbucket.org/worldsayshi/master_thesis-all. Online; accessed 2013-02-28

59. CAPRA, Rob and ET AL. Symposium on Human-Computer Interaction and Information Retrieval. In: [online]. Available from: https://sites.google.com/site/hcirworkshop/. Online; accessed 2013-02-28

60. AANEN, Steven, NEDERSTIGT, Lennart, VANDIĆ, Damir and FRĂSINCAR, Flavius. SCHEMA-an algorithm for automated product taxonomy mapping in e-commerce. In: *The Semantic Web: Research and Applications*. 2012, pp. 300–314.

61. YU, C., SUN, W., DAO, S. and KEIRSEY, D. Determining relationships among attributes for interoperability of multi-database systems. In: *Interoperability in Multidatabase Systems, 1991. IMS'91. Proceedings., First International Workshop on*. S.l.: IEEE, 1991. pp. 251–257.

62. WAUTHIER, F. L., JOJIC, N. and JORDAN, M. I. Active Spectral Clustering via Iterative Uncertainty Reduction. In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. S.l.: ACM, 2012. pp. 1339–1347.