
ALWIS

A Visualization Tool for Concept Based Retrieval Schemes
Design and Implementation

Bachelor thesis

Daniel Fischer
13th March 2006



Max Planck Institut Informatik, Saarbrücken
Abteilung 1, Algorithms and Complexity

Betreuer: Holger Bast

Abstract

Comparing and evaluating the performance of concept based retrieval schemes is notoriously difficult and of increasing practical importance. To aid the evaluation process we developed a visualization tool which puts particular emphasis on the comparison of single queries. This tool directly visualizes a generic theoretical model for concept based retrieval schemes. In this paper we derive the graphical user interface from the model and present the design of the implementation. A notable feature of the implementation is its extensibility with regard to additional concept based retrieval schemes.

Contents

1	Introduction	1
2	Results	3
3	The Graphical User Interface	5
3.1	The main window	5
3.1.1	Three views	7
3.1.2	Elementary queries	7
3.1.3	Combined queries	8
3.1.4	Multiple models	8
3.1.5	Weighted and unweighted queries	9
3.1.6	Most relevant documents for a term query	9
3.2	The comparison window	10
3.3	The measurement wizard	11
4	Internal structure	13
4.1	Separation of model calculation and GUI	13
4.2	Separation of query execution and GUI	14
4.2.1	Various options	15
4.2.2	Interactive loading issue	16
4.2.3	Conclusion	16
4.3	Separation of measurement and GUI	16
5	Implementation	19
5.1	Calculation of models	19
5.1.1	Stream concept	19
5.1.2	Preprocessing of corpora	19
5.1.3	The actual calculation of models	21
5.2	Query engine	22
5.3	Measurement devices	23
5.4	Graphical user interface	24
6	Conclusion and future work	25
	Acknowledgments	27
	List of Figures	29

Chapter 1

Introduction

A standard problem in computer science is the so called retrieval problem: How to search for a set of relevant documents given a query (that is a set of terms). Most of the time one is not actually interested in all relevant documents but just the “most relevant” document(s). Therefore the result should also include a number denoting its relevance. Which can then be used to sort the resulting set. This resulting sequence is also known as a ranking.

The most obvious approach to generate such a ranking is just to search for documents containing the terms mentioned in the query. The relevance of a document is then directly proportional to the number of query terms which are actually contained in the document.

This can be implemented very easily by an index, denoting which documents a term is contained in. Unfortunately this approach cannot cope with two phenomena, which occur very often in natural language:

- Polysems, that is words which have more than one meaning. For instance the word trunk has several meanings: a suitcase, the trunk of an elephant, the torso as well as a bole.
- Synonyms, that is several words which have the same meaning. For example the words: performer, entertainer, artist and player can all be used to denote the concept of an actor.

These phenomena can have a severe impact on the performance of a retrieval algorithm. The users of such an algorithm are normally not interested in specific words but in their meaning. But as seen above a word might actually have several meanings and there might also be several words for the same meaning. Therefore we need a way to identify or at least guess / approximate the meaning of both queries and documents.

One popular class of algorithms used to solve this problem are the so called concept based retrieval schemes. They introduce a new way to describe documents, in which documents are no longer described by the terms they contain, but rather by the concepts they belong to.

In general these algorithms consist of two phases. In the first phase the concepts contained in the corpus (the set of known documents) are identified. Intuitively a *concept* can be understood as a topic common to several documents in the corpus. The documents are then recast into a so called *concept space* representation. In the common *term space* representation a document is equal to its terms – each term weighted by its number of occurrences. In concept space each document is equal to the concepts “contained” in it, each concept weighted according to its relevance. Thereby we have now created a fuzzy clustering of the documents into their concepts. Note that these algorithms perform unsupervised clustering, in particular they do *not* need a labeled corpus.

In the second (query) phase the query is cast into concept space as well. The ranking can then be created by a simple comparison in concept space of the “query document” with all other documents.

Typically the performance of a retrieval scheme is measured by comparing the results of several queries with predetermined optimal results. The resulting average performance gives a good hint on the overall performance of an algorithm. But to understand why an algorithm performs well (or not) for certain queries it is necessary to actually look at the result of the query itself and compare it to the results of other algorithms on the same query. Ideally one would be able to compare the query and its results in term as well as in concept space.

Structure of the paper

First we will give a short overview of the results of our work. Then we will present the graphical user interface (GUI) we developed. We will talk about the individual components of the GUI and how they are used. After that we will go into technical details. Especially we will motivate what algorithmic parts were sourced out from from the GUI for which reasons. Finally we will talk about the actual implementation of all tools belonging to the ALWIS suite.

Chapter 2

Results

Therefore we developed a visualization tool which allows us to compare different concept based retrieval schemes. Unlike previous attempts we turned our attention on comparing the results of a *single, specific* query on several different clusterings of the same corpus directly.

The notion of a concept is made explicit in our visualization tool. Therefore we can display an approximation of the concept space introduced by a concept based retrieval scheme in our visualization tool.

In preparation of our visualization tool we introduced the following theoretical foundations:

- We developed a generic model which unifies the different concept based retrieval schemes. Furthermore we generalized queries. Thereby we allow *nine* kinds of queries instead of only the classic one which retrieves the most relevant documents for a set of terms.
- We showed how to map concept based retrieval schemes into our generic model. We provide mappings for LSI, PLSI and NMF. Similar algorithms can easily be mapped in an analogous way.
- By the example of Spectral Clustering we showed that it is even possible to use plain clustering algorithms – which do not allow querying by themselves – for information retrieval.

We developed a suite of applications which is called ALWIS. As mentioned before concept based retrieval schemes are two phased algorithms. Our visualization tool naturally only visualizes the query phase. Nevertheless we also provided all necessary tools needed in the first phase. In particular we provide tools to prepare a corpus. All in all it consists of the following components:

- We provide five tools for corpus processing. This includes e.g. document filtering and stemming. Stream processing via Unix pipes is supported where appropriate. Altogether these tools consist of roughly 1900 lines of code.
- Three tools for calculating models on the basis of corpora are provided as well – for LSI, PLSI and Spectral Clustering. In total there are roughly 3100 lines of code for this tools.
- Query visualization is done in an intuitive graphical user interface which allows mouse control as well as keyboard control. It took four prototypes to reach this level of usability. The GUI consists of roughly 19300 lines of code.
- The graphical user engine uses two supplementary tools: the query engine and the precision recall measurement device. Both together consist of roughly 1700 lines of code.
- The usage of the applications as well as technical details like file formats and protocols are described in the reference manual. The manual is provided in HTML and PDF format and consists of roughly 55 pages.

One important issue in the development of the ALWIS suite was making the applications cross-platform. Especially process creation and inter-process communication with Unix pipes in the presence of a graphical user interface were problematic in this context. So far the applications have been tested on Linux and on Windows. Porting to other POSIX compliant platforms supported by the `wxWidgets` GUI library should be very easy. For Windows we provide a precompiled binary package coming with a full featured installer.

Due to the size of this project the work was split between my colleague Benedikt Grundmann [5] and myself. In this paper I present the results of my part which focuses on the graphical user interface and its implementation.

Chapter 3

The Graphical User Interface

As denoted by my colleague Benedikt Grundmann [5] concept based retrieval schemes are two phase algorithms. They consist of

- a clustering phase in which a model is calculated and
- a query phase in which a calculated model is used to perform queries.

This graphical user interface (GUI) helps you to perform queries and to compare the results from these queries, i.e. the GUI completely focuses on the query phase of concept based retrieval schemes. It does *not* help you to calculate models. In the following chapters we will present the individual parts of the GUI.

3.1 The main window

A typical search engine interface consists just of a text box in which the query is entered and a window in which the relevant documents are listed. But in order to allow a deeper insight into cluster based retrieval schemes our generic model, as introduced in [5], removes this layer of abstraction. Therefore we have direct access to three entities: documents, terms and concepts. Furthermore there are six query kinds instead of only one: instead of only being able to map a term query to document results you can map from any of the three entities to any of the remaining two entities. It is quite clear that the typical search engine interface is not powerful enough to meet our generic model.

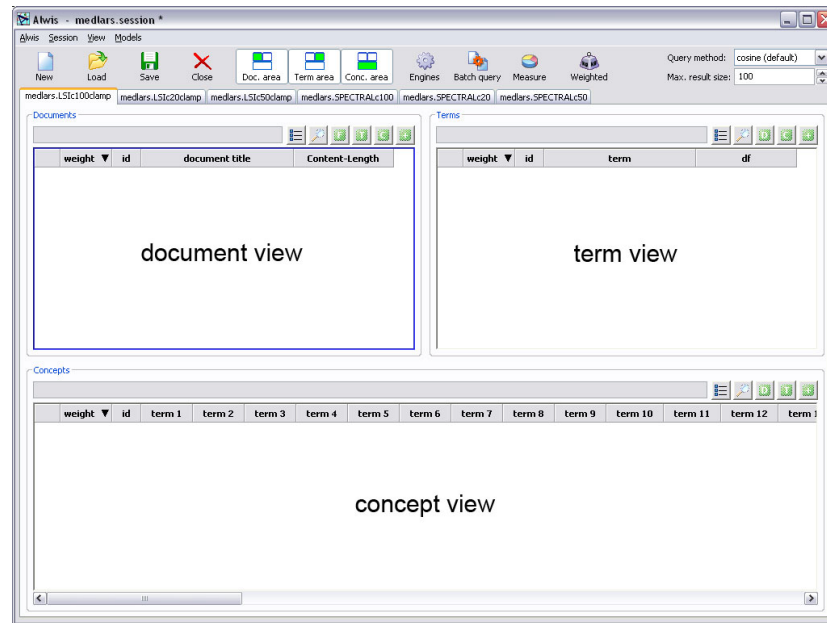


Figure 3.1: The main window of ALWIS

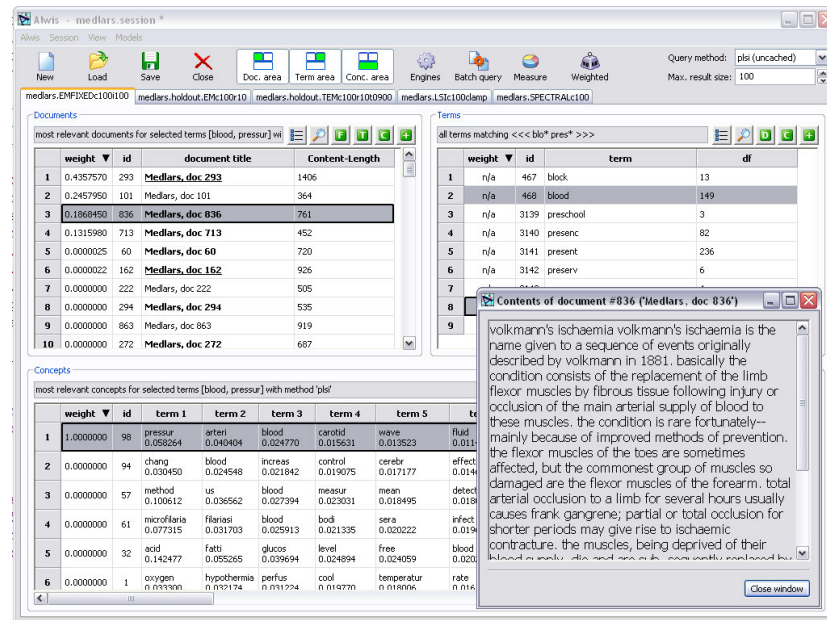


Figure 3.2: The three views showing some data from the medlars collection and a window showing the contents of a document

3.1.1 Three views

As you can see in figure 3.1 most space of the main window is taken by three views directly corresponding to the three entities in the generic model: a document view, a term view and a concept view. Each view shows a list of items from the respective entity.

Internally these items are just numbers, namely row or column indices for the matrices a model consists of, with optional weights. Since displaying just these ids would not be very convenient we need to resolve them. The document space as well as the term space are well known, therefore we can provide a usable description for document items and term items. Document items are annotated with the document's title and the document's length. On double-clicking a document item the actual contents of that document are shown. Term items are annotated with the term itself and its document frequency.

Unfortunately cluster based retrieval schemes do *not* provide any semantic description of what a specific concept is. We can only use the implicit information stored in the model to annotate concept items. Therefore concepts are annotated with the most relevant terms for the respective concept. In figure 3.2 you can see the three views showing some data from the medlars collection.

3.1.2 Elementary queries

Performing a query is equivalent to mapping items from one entity, i.e. one view, to one of the remaining entities, i.e. one of the two remaining views. In practice this is done by selecting items in one view and clicking the appropriate button near one of the remaining views. In detail there are six elementary queries you can perform:

- **most relevant documents for the selected terms**
After selecting one or more terms in the term view click the **T** button near the document view. This is equivalent to pressing **Ctrl+D** while the term view has the keyboard focus.
- **most relevant documents for the selected concepts**
After selecting one or more concepts in the concept view click the **C** button near the document view. This is equivalent to pressing **Ctrl+D** while the concept view has the keyboard focus.
- **most relevant terms for the selected documents**
After selecting one or more documents in the document view click the **D** button near the term view. This is equivalent to pressing **Ctrl+T** while the document view has the keyboard focus.
- **most relevant terms for the selected concepts**
After selecting one or more concepts in the concept view click the **C**

button near the term view. This is equivalent to pressing `Ctrl+T` while the concept view has the keyboard focus.

- **most relevant concepts for the selected documents**

After selecting one or more documents in the document view click the **D** button near the concept view. This is equivalent to pressing `Ctrl+C` while the document view has the keyboard focus.

- **most relevant concepts for the selected terms**

After selecting one or more terms in the term view click the **T** button near the concept view. This is equivalent to pressing `Ctrl+C` while the term view has the keyboard focus.

3.1.3 Combined queries

Furthermore there are so called combined queries. A combined query actually consists of two elementary queries whose results are combined by multiplying the returned weights of each returned item. That means in detail:

- only items contained in both query results are contained in the final query result
- only items ranked high in *both* query results will be ranked high in the final query result

You can perform a combined query by clicking on the **+** button near one of the views. If you click e.g. on the **+** button near the document view the most relevant documents for the selected terms and the most relevant documents for the selected concepts will be queried. The resulting (combined) query result is shown in the document view and can be interpreted as the most relevant documents for the selected terms while limiting the meaning of these terms to that of the selected concepts.

3.1.4 Multiple models

This tool has been developed to allow the comparison of different cluster based retrieval schemes. In terms of our generic model this means that we need to compare different models. But what does comparing models actually mean? The most intuitive comparison is to perform the same query on different models and comparing the query results. Therefore we need to concurrently perform queries on different models. Of course switching between the models should require as less effort as possible.

Screen space is technically limited. Therefore it is virtually impossible to show the set of three views each for many models in parallel. Regarding special

display devices like beamers it is often even not possible to show two models in parallel. Therefore we have chosen to utilize the tabbed window paradigm. Thereby the working area of the window is enlarged by stacking multiple layers of views. There is one such layer for each model, each containing a document view, a term view and a concept view for the appropriate model. You can easily switch between the layers/models by clicking on the appropriate tab.

3.1.5 Weighted and unweighted queries


By default all queries are performed unweighted. That means that the selected items will all have the same weight. The weights of the selected items will always sum up to 1. The content of the weight column is *not* considered. Most often this is the wanted behavior.

Sometimes you may want to take the weights from the weight column into account. Each selected item is weighted with the appropriate value from the weight column but the values are normalized so that the weights of all selected items sum up to 1. You can perform weighted queries by activating the weighted query mode by clicking the appropriate button in the tool bar. After the query has been performed the weighted query mode is disabled automatically.

3.1.6 Most relevant documents for a term query

Usually one is interested in the most relevant documents for a query consisting of some terms. This kind of query is commonly performed in a special way by cluster based retrieval schemes. First the term query is mapped into concept space and then the most relevant documents for this concept space representation of the query are queried. This is different from the elementary query “most relevant documents for the selected terms” from above! The elementary query described above directly maps from term space to document space instead of mapping to concept space first!

We have made these individual steps explicit in our generic model. Please refer to the work of Benedikt Grundmann [5] for more information on this. But of course this special procedure can be done in the GUI as well by performing the following steps:

- select the appropriate terms for which you are interested in the most relevant documents
- query the most relevant concepts for the selected terms by clicking on the  button near the concept view or by pressing **Ctrl+C** while the term view has the keyboard focus (this step is the actual folding in of the term query into the concept space)

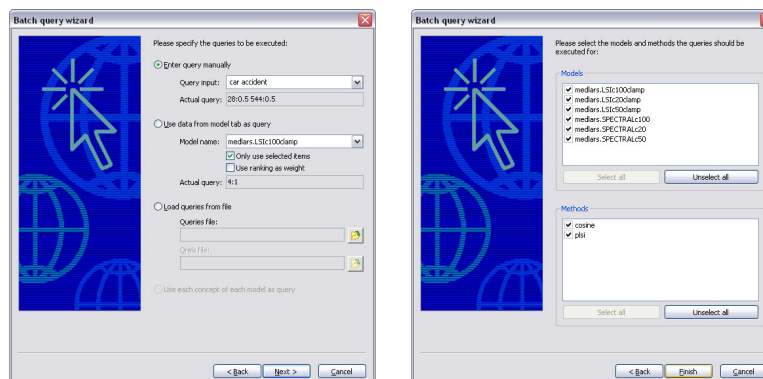


Figure 3.3: Two pages from the batch query wizard

- activate the weighted query modus by clicking the appropriate button in the tool bar or by pressing **Ctrl+W**
- select all concepts in the concept view by setting the keyboard focus to the concept view (e.g. click somewhere in the concept view) and pressing **Ctrl+A**
- query the most relevant documents for the selected concepts by clicking on the **C** button near the document view or by pressing **Ctrl+D** while the concept view has the keyboard focus

For convenience reasons we added a special button **F** which encapsulates these five steps into one step. After having selected one or more terms in the term view just click on this button which is located near the document view. The concept view is updated in order to give you a hint how your term query is mapped into concept space.

3.2 The comparison window

Switching between different models in order to find (minor) differences in query results can be quite tedious. Therefore we provide a comparison window which directly sets multiple query results in contrast to each other. Using the batch query wizard, see figure 3.3, you can specify which queries should be executed for which models using which query methods. The query results are presented in a new window which is depicted in figure 3.4.

In this window the different query results are stacked in columns which allows an easy comparison. If you select one or more items in one query result the appropriate items in all other query results are highlighted, too. This makes locating corresponding items even more easy. For a better overview you can temporarily

The screenshot shows the 'Atwis comparison window' with three query result tables. The left table is titled 'Query 'crystallin len vertebr includ human Model 'medlars.LStc100clamp' Method 'cosine' (folding in)'. The middle table is titled 'Query 'crystallin len vertebr includ human Model 'medlars.LStc100clamp' Method 'cosine''. The right table is titled 'Query 'crystallin len vertebr includ human Model 'medlars.LStc100clamp' Method 'pls' (folding in)'. Each table has columns for '#', 'weight', 'id', and 'document title'. The results are sorted by weight in descending order.

#	weight	id	document title	#	weight	id	document title	#	weight	id	document title
1	0.0020721	1	Medlars, doc 1	1	0.0020721	1	Medlars, doc 1	1	0.2757300	652	Medlars, doc 652
2	0.0019069	633	Medlars, doc 633	2	0.0019069	633	Medlars, doc 633	2	0.1712780	630	Medlars, doc 630
3	0.0016885	787	Medlars, doc 787	3	0.0016885	787	Medlars, doc 787	3	0.1425020	658	Medlars, doc 658
4	0.0016418	783	Medlars, doc 783	4	0.0016418	783	Medlars, doc 783	4	0.1373740	633	Medlars, doc 633
5	0.0016134	612	Medlars, doc 612	5	0.0016134	612	Medlars, doc 612	5	0.0800783	737	Medlars, doc 737
6	0.0016016	34	Medlars, doc 34	6	0.0016016	34	Medlars, doc 34	6	0.0567235	864	Medlars, doc 864
7	0.0015993	121	Medlars, doc 121	7	0.0015993	121	Medlars, doc 121	7	0.0475187	586	Medlars, doc 586
8	0.0015965	545	Medlars, doc 545	8	0.0015965	545	Medlars, doc 545	8	0.0448859	677	Medlars, doc 677
9	0.0015858	864	Medlars, doc 864	9	0.0015858	864	Medlars, doc 864	9	0.0200274	729	Medlars, doc 729
10	0.0015820	538	Medlars, doc 538	10	0.0015820	538	Medlars, doc 538	10	0.0160337	888	Medlars, doc 888
11	0.0015486	678	Medlars, doc 678	11	0.0015486	678	Medlars, doc 678	11	0.0055735	678	Medlars, doc 678
12	0.0015036	762	Medlars, doc 762	12	0.0015036	762	Medlars, doc 762	12	0.0016110	756	Medlars, doc 756
13	0.0014983	575	Medlars, doc 575	13	0.0014983	575	Medlars, doc 575	13	0.0003087	558	Medlars, doc 558
14	0.0014959	78	Medlars, doc 78	14	0.0014959	78	Medlars, doc 78	14	0.0001819	447	Medlars, doc 447
15	0.0014839	310	Medlars, doc 310	15	0.0014839	310	Medlars, doc 310	15	0.0000592	757	Medlars, doc 757
16	0.0014720	186	Medlars, doc 186	16	0.0014720	186	Medlars, doc 186	16	0.0000336	593	Medlars, doc 593
17	0.0014508	227	Medlars, doc 227	17	0.0014508	227	Medlars, doc 227	17	0.0000244	531	Medlars, doc 531
18	0.0014500	658	Medlars, doc 658	18	0.0014500	658	Medlars, doc 658	18	0.0000194	679	Medlars, doc 679
19	0.0014434	325	Medlars, doc 325	19	0.0014434	325	Medlars, doc 325	19	0.0000123	787	Medlars, doc 787
20	0.0014380	7	Medlars, doc 7	20	0.0014380	7	Medlars, doc 7	20	0.0000086	575	Medlars, doc 575
21	0.0014289	492	Medlars, doc 492	21	0.0014289	492	Medlars, doc 492	21	0.0000036	713	Medlars, doc 713
22	0.0014220	585	Medlars, doc 585	22	0.0014220	585	Medlars, doc 585	22	0.0000013	471	Medlars, doc 471
23	0.0014203	96	Medlars, doc 96	23	0.0014203	96	Medlars, doc 96	23	0.0000005	837	Medlars, doc 837
24	0.0014135	580	Medlars, doc 580	24	0.0014135	580	Medlars, doc 580	24	0.0000002	675	Medlars, doc 675

Figure 3.4: The comparison window

hide the query results from specific queries, models or query methods using the check box lists at the left edge of the window.

3.3 The measurement wizard

There are some methods to (semi-)automatically measure models. For instance you can draw a precision recall curve which denotes the precision of query results at different recall levels. For more details on measurement methods please refer to [5].

The measurement wizard is very similar to the batch query wizard from above. After specifying the measurement device to use you can specify which queries should be executed for which models using which query methods. After the queries have been executed the results, i.e. in this case the precision recall curve, are presented in a new window.

Chapter 4

Internal structure

4.1 Separation of model calculation and GUI

As denoted by my colleague Benedikt Grundmann [5] concept based retrieval schemes are two phase algorithms. As we have stated earlier the graphical user interface *completely* focuses on the second phase, i.e. the query phase. The first phase, i.e. the clustering phase, in which the models are calculated, has to be done externally. But what are the reasons for this separation?

The obvious, monolithic approach to the development for a visualization tool like ALWIS would integrate both phases into one application. But such an approach would have several drawbacks:

- The extension of a monolithic application to new retrieval schemes would require changing potentially large portions of the application's source code and recompiling the whole application. This would make the extension to new algorithms more difficult and more error prone.
- Any new retrieval scheme would have to be implemented in the programming language the monolithic application is written in. Alternatively one could use bindings of other programming languages to the one of the monolithic application. In both cases the implementation of new retrieval schemes potentially gets more troublesome.
- Most clustering algorithms are computational intense and time consuming. Integrating the calculation of models into a monolithic graphical user interface would complicate or even prevent the calculation of models using remote computation servers. Taking into account that e.g. the computation of a PLSI model on a medium sized corpus can take up to several days the integration of the calculation into a GUI is obviously no good idea.

Therefore we decided to strictly separate the two phases of concept based retrieval schemes in the implementation as well. This separation is made possible by our generic model. For more details on the generic model please refer to the thesis of Benedikt Grundmann [5].

The calculation of models is implemented as external command line tools without a graphical user interface. They have to be invoked using a shell. This approach has some important advantages:

- The separation of the calculation of models from the query visualization allows us to use *any* tool to calculate models. We can use well-established suites like Matlab as well as other programs written in any programming language for this purpose. This ensures maximum flexibility.
- By implementing the calculation of models as command line tools using remote computation servers for the calculation of models gets very easy. Simply establish a **SSH** connection to the computation server, create a new **screen** session and start the calculation of a model. You can now detach the **screen** session and close the **SSH** connection at any time and reconnect later to see the results.
- Knowing about the computational intensity of the calculation of models we would like to have the possibility to store models calculated once for later use. This feature of loadable and storable models is used for the model exchange between calculation and visualization as well. Therefore we get the separation between of the calculation from the visualization “for free”.

This approach has only one slight disadvantage. Model computation tools without graphical user interface are slightly less convenient since they require the user to use a shell. But this is acceptable since ALWIS is targeted at researchers which should be familiar with using a shell. Of course one could think of an optional graphical front-end for these tools, but it is not provided in the ALWIS package. Fortunately a graphical front-end interfacing to the command line tools would not break the advantages stated above.

4.2 Separation of query execution and GUI

As denoted by my colleague Benedikt Grundmann [5] there can be several so called query methods. A query method is an algorithm which is used to calculate a query result for a given query on the basis of a given model. The most basic query method is the cosine comparison which just uses the dot products of the query vector with the appropriate vectors from the model to calculate the query result. Of course we would like to choose freely among all available query methods. In the following we will refer to the implementation of query methods by the notion “query engine” (QE).

We can think of several possibilities of how to implement the different query methods:

1. all query methods are integrated into the visualization tool
2. one external process per model and query method
3. one external process per model, each utilized for every query method
4. exactly one external process, utilized for each model as well for each query method

Each of these options has its own advantages and disadvantages. In the following we will further explain the options and discuss their pros and cons.

4.2.1 Various options

Option 1

Integrating all query methods into the visualization tool itself is clearly not an option. There would be the same problems as stated above, please refer to section 4.1. This approach would break the flexibility of simply adding new cluster based retrieval schemes to ALWIS which is one of our main goals.

Option 2

Assuming we support n different query methods we would need n different applications, each implementing one query method. The visualization tool would start one instance of each of these applications for each model the user is currently visualizing. While this approach is the theoretically most beautiful one there is one major drawback. Each query engine needs access to the model data which would have to be stored in memory. Since we have n different query engines in memory for *each* model this means that we would have n identical copies of the model data in memory. Since the model data is quite memory intense this would put unnecessary limits to the number of models and query methods the user could investigate in parallel. Unfortunately sharing the memory containing the model data between the separate processes does not work in general – there would arise cross-platform problems and problems with query engines written in different programming languages.

Option 3

In this approach we would need to implement exactly one query engine application. The visualization tool would start exactly one instance of this query engine for every model the user is currently visualizing. Since different models imply different model data we would not hold duplicate copies of the same data in memory. Although we separate the query methods from the visualization

we would have to provide *one* query engine capable of performing *every* query method. This slightly impairs the flexibility.

Option 4

This option is almost identical to option 3 despite the fact there is exactly one global instance of a query engine. This global instance would be responsible for performing queries on any model using any query method. Again we would need to implement exactly one query engine application. But this time we would need to cope with any number of models within this application. It is quite obvious that this would complicate the implementation of the query engine. Therefore this option would further impair the flexibility without gaining any benefit over option 3.

4.2.2 Interactive loading issue

One could think about interactive loading – either of the model data or even the complete query engine process including the data – as a solution to some problems stated above. After doing some benchmarking we had to recognize that is no option either. Interactively forking the process and loading the model data would introduce non-neglectable delays into the visualization process which renders this strategy useless.

4.2.3 Conclusion

Unfortunately there is currently no solution meeting all theoretical as well as all practical requirements. Since our goal is a working implementation we have to focus on the practical requirements. Therefore we decided to chose option 3. This is a practical compromise which only slightly restricts our goal of flexibility. Here we do *not* have any convenience restrictions since the user never uses the query engine application directly but only through the graphical user interface . In figure 4.1 you see a sketch of the implemented solution.

4.3 Separation of measurement and GUI

As seen in the thesis of Benedikt Grundmann [5] there are some methods to – at least semi-automatically – evaluate the quality of models and query execution. This can be done using only the model matrices, like e.g. for calculating the relative entropy, or using the results of the execution of several queries, what e.g. is done when calculating a precision recall curve.

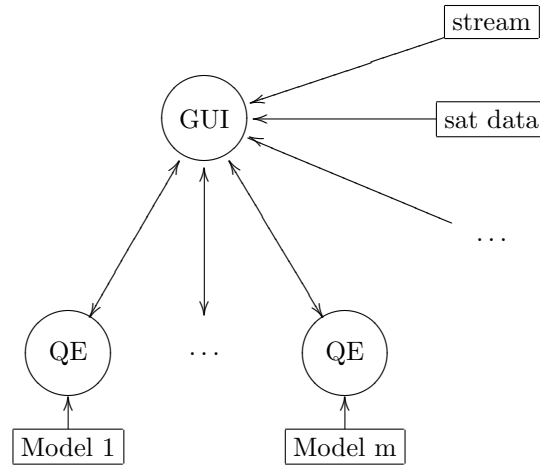


Figure 4.1: Separation of the query engine from the GUI

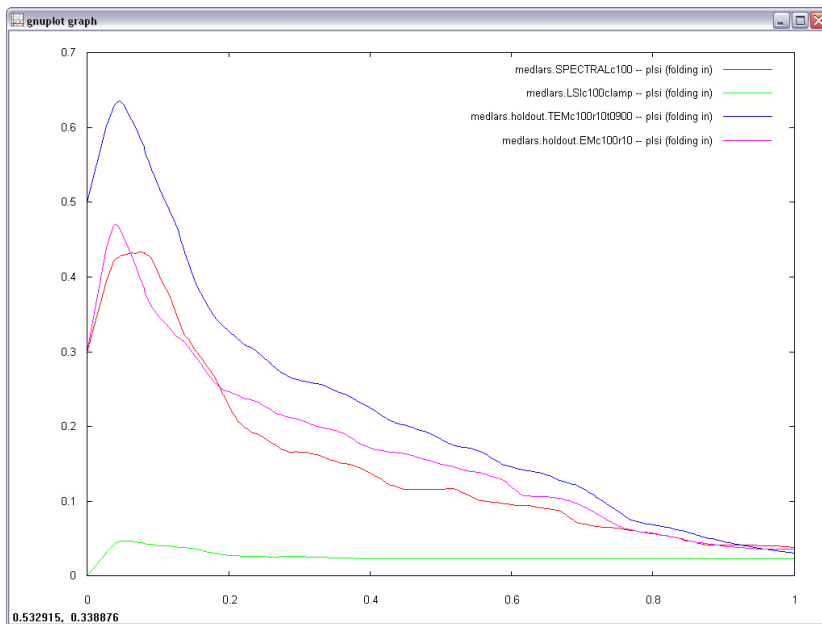


Figure 4.2: Output of the precision recall measurement device

Once again we decided to implement measurement devices as external applications and not as integral part of the graphical user interface. If the respective measurement device requires query results the GUI does the actual query execution for the measurement device. All eventually necessary parameters as well as the query results are passed via the command line or via files to the external measurement application. It is up to the measurement device how to present its result to the user. It is free to output its results to the console or to present its results in an own graphical user interface. For example the implemented precision recall measurement device presents the calculated curve in a new gnuplot window as depicted in figure 4.2.

This approach makes adding new measurement devices very easy. You are completely free in choosing an appropriate programming language for implementing the measurement device. Furthermore you do not need to bother to execute the queries since this is done by the graphical user interface. The only technical infrastructure needed in the new measurement device is simple command line parsing and optionally parsing files with a very simple text format. Almost every programming language provides sophisticated means for such tasks.

Chapter 5

Implementation

5.1 Calculation of models

As one can see in the work of Benedikt Grundmann [5] the actual calculation of models requires some preprocessing of the corpora. In the generic model the calculation of models is based upon a document term matrix (DTM) which is a sparse matrix representing the preprocessed contents of a corpus.

5.1.1 Stream concept

In general a corpus is just a bunch of files. Since we focus on text corpora all these files are (or should at least be) text files. Most time we want to treat all files contained in a corpus the same way. In this context coping with all the single files individually is not very convenient.

Therefore we adapted the stream concept proposed in [1] for our needs. A stream represents a corpus by bundling all files belonging to this corpus into one file. Despite the actual file contents the stream contains some additional meta information like a document title for every file in the stream. In particular this approach enables us to implement filter applications for whole corpora/streams using the concept of Unix pipes.

5.1.2 Preprocessing of corpora

In the following we will assume a corpus consisting of several plain text files. Other file types like HTML, Postscript or PDF should be converted to plain text

beforehand using third party tools. In general we will have to do the following preprocessing steps:

- merging plain text files into a stream,
- filtering out unwanted characters (term delimiter characters),
- filtering out unwanted terms (stop words, see [8]),
- stemming of terms (see [4]) and
- building a document term matrix (DTM) and optionally a hold-out matrix from the filtered document stream

Depending on the origin and the state of the stream some steps stated above may be unnecessary or vice versa some additional steps may be necessary. For details on how important a proper preprocessing of corpora is and how it can improve the quality of the calculated models please refer to [5].

It is quite obvious that the single preprocessing steps stated above are independent from each other. In order to maintain maximum flexibility we would like to have a mechanism which allows us to reorder or remove preprocessing steps as well as adding further preprocessing steps.

This is achieved by implementing the single preprocessing steps as individual console applications. Each such application reads one or more files and outputs one or more files. We actually implemented four such applications using the programming language C++:

- `cat2stream` reads multiple text files and outputs a stream containing these files
- `filterstream` reads a stream, removes unwanted characters and terms from the contained documents and outputs the filtered stream
- `stemstream` reads a stream, stems all terms in the contained documents and outputs the filtered stream
- `stream2matrix` reads a stream and outputs a document term matrix and an optional hold-out matrix as well as satellite data (consisting of a document list and a term list)

The document term matrix (DTM) as well as the optional hold-out matrix is used to calculate models later on. The DTM represents the contents of the corpus in form of a sparse matrix stating the number of occurrences of each term in each document. The optional hold-out matrix contains data which is used to measure the quality of a model while it is iteratively trained (so far only the PLSI algorithm uses this). The satellite data created by `stream2matrix`

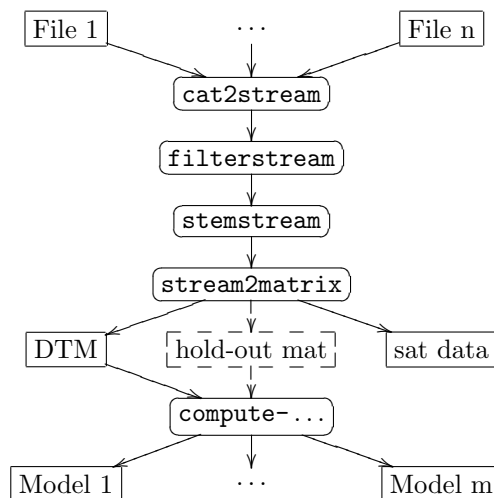


Figure 5.1: One possible way from a corpus to models

is only used in the graphical user interface for visualization purposes and is not needed to calculate a model. A possible processing chain ranging from the original corpus to the finally calculated models is depicted in figure 5.1.

Most of the tools support communication via Unix pipes. This applies to all tools reading only one input file and/or writing only one output file. In our implementation we fixed some cross-platform issues which prevented piping from working on Windows. It may be worthwhile to have a look into our source code [3] before trying to implement an additional console application for the use in the processing chain.

5.1.3 The actual calculation of models

As stated in our generic model [5] the calculation of models based upon a corpus only utilizes the DTM matrix (and an optional hold-out matrix) which represents the contents of the corpus.

There is one command line tool for the model calculation for each cluster based retrieval scheme. We actually implemented three such model calculation tools:

- `compute-lsi` computes a LSI model from a given document term matrix according to the LSI algorithm described in [2], this tool is implemented in C++
- `compute-plsi` computes a PLSI model from a given document term matrix and a given hold-out matrix according to the PLSI algorithm described in [6], this tool is implemented in OCaml

- `compute-spectral` computes a Spectral Clustering model from a given document term matrix according to the Spectral Clustering algorithm described in [7], this tool is implemented in C++

Each of these tools outputs two files: a `pzd` matrix and a `pwz` matrix. These two files form a model as described in our generic model [5]. The model files are used in the query phase to execute the queries.

We would like to emphasize one important aspect of our choice of the used programming languages. We consciously did not chose to use Matlab or any other well-established math suite for the calculation of models. Although such suites provide almost every algebraic function we will ever need for the calculation of models one major problem persists. Not all people have access to a suitably licensed copy of Matlab. When relying on Matlab we would prevent people without access to Matlab from calculating their own models. Of course this applies to other commercial suites as well.

5.2 Query engine

As we stated in section 4.2 we decided to separate the actual query executing from the graphical user interface. Furthermore – as a necessary practical compromise – we decided to implement *one* query engine application which is able to execute *all* query methods. The graphical user interface forks *one* query engine process for every model the user wants to visualize.

Communication between the graphical user interface and the query engine is done via Unix pipes. The graphical user interface prints all queries which should be executed to standard input of the query engine process. The query engine then prints the query results to its standard output which is read by the graphical user interface. The communication is actually done using a very simple text protocol.

The query engine is implemented in C++ in order to ease the implementation of pipe based communication.

It is important to know that the query engine has to implement only the six elementary queries

- most relevant documents for a term query,
- most relevant documents for a concept query,
- most relevant terms for a document query,
- most relevant terms for a concept query,

- most relevant concepts for a document query and
- most relevant concepts for a term query.

All other queries, i.e. the combined queries as well as querying with folding in, are performed by the graphical user interface by combining the results from multiple elementary queries. This additionally eases the implementation and extension of the query engine.

There can be multiple query methods in the query engine. As stated in our generic model [5] a model consists of a **pzd** matrix and a **pwz** matrix. A query is just a vector of weighted documents, terms or concepts. A query method is an algorithm which describes how the query result is actually derived from a given model and a given query. The query engine has to support any combination of elementary query and query method. Currently there are two query methods implemented in the query engine:

- cosine comparison and
- generalized PLSI style folding in

Please refer to the work of my colleague Benedikt Grundmann [5] for details on this query methods.

5.3 Measurement devices

As we stated in section 4.3 we decided to separate the measurement of models from the graphical user interface. All necessary data is provided to the measurement device. Especially it does not have to bother with executing queries since the query execution is managed by the graphical user interface.

We actually implemented one measurement device calculating a precision recall curve. The application is written in OCaml. It calculates the precision recall curve using the query results which it got as input and uses gnuplot to actually plot the curve. The plotted curve is presented in a new window.

As it is commonly known precision and recall are defined as follows:

$$\text{precision} = \frac{\text{retrel}}{\text{returned}}$$

$$\text{recall} = \frac{\text{retrel}}{\text{relevant}}$$

retrel is the number of *relevant* documents contained in a query result, **returned** is the *total* size of the query result and **relevant** is the number of *all* relevant documents the corpus contains.

The set of relevant documents in a corpus has to be determined by hand. Obviously there is no automatic way to this – if such a method would exist it would be the perfect search algorithm. Fortunately there are test collections which supply a set of queries and the set of relevant documents for each of these queries.

In order to get a precision recall curve we analyze the precision at different recall levels. By using the recall level as x coordinate and the appropriate precision as y coordinate we get points forming the precision recall curve.

Such a curve would not be very representative for a retrieval scheme if we would only consider *one* query – one could think of a fake retrieval scheme specialized to exactly this query. Therefore we consider as many queries as possible, each with its own set of relevant documents. By averaging the resulting curves we get a smoother and more representative precision recall curve for the analyzed retrieval scheme.

For more details on the implementation, especially on the averaging of the precision recall curves from the individual query results, please refer to the source code [3]. In figure 4.2 a sample output of this measurement device is depicted.

5.4 Graphical user interface

In order to maximize portability we have chosen to implement the graphical user interface in C++. In particular there is a reasonably well tested and stable C++ cross-platform toolkit for the development of graphical user interfaces, namely wxWidgets. wxWidgets has the advantage of using native controls of each operating system so that the user gets an application with a familiar look and feel.

The graphical user interface transparently handles forking and terminating of external processes needed as well as the communication with them. For example for every model which is visualized a new instance of the query engine is forked automatically. The experienced user has minimal control over the external processes. He can (re)start or terminate a query engine and has the ability to change the command line arguments of the query engine. Hiding most of the technical stuff from the user makes our tool accessible to a larger audience.

The actual implementation of the graphical user interface in principle consists only of plain GUI code and some technical stuff like file parsing and interprocess communication. All other tasks, like model calculation, query execution and measurement, are sourced out into external applications. The main advantage of this is that the algorithmic code of the external applications is not mixed with GUI code.

Chapter 6

Conclusion and future work

Together with my colleague Daniel Fischer I implemented a visualization tool which puts emphasis on the comparison of single specific queries across several concept based retrieval schemes.

To facilitate that we have shown how to map different concept based retrieval schemes into the generic model. This model introduces the notion of a generalized query. The generalized query has proven itself to be helpful in understanding the performance of a concept based retrieval scheme on a single query. For certain applications it might be advantageous to provide the power of the generic model to the end user. For example a movie database might want to provide means to list related movies.

We are convinced of the usefulness of our tool, nevertheless we are interested in reports of using ALWIS. In particular we would like to hear about additional mappings of other concept based retrieval schemes into the generic model.

We hope that our tool will help in understanding the performance of concept based retrieval schemes. It might be interesting to compare the performance of the same clustering using different kinds of query functions.

Acknowledgments

Our supervisors Holger Bast and his colleague Ingmar Weber were very helpful during the development of ALWIS. They tested the different prototypes of the graphical user interface. Both are actually working in the field of concept based retrieval schemes and as such they are future users of our program. This tight feedback loop resulted in a lot of constructive criticism and useful suggestions.

As mentioned before we developed ALWIS in a team of two. I would like to thank my colleague Benedikt Grundmann for helping me with his algorithmic knowledge and his very detailed knowledge of different programming languages. This saved me quite some headache. And of course it was quite important to have someone listening to my problems and searching for my bugs.

I also want to thank one of my fellow students, who was always ready to provide distractions when we needed them. She will know who we are talking about.

Finally, but most importantly, I would like to thank my family for their constant support over all these years – especially for enabling me to study computer science.

List of Figures

3.1	The main window of ALWIS	6
3.2	The three views showing some data from the medlars collection and a window showing the contents of a document	6
3.3	Two pages from the batch query wizard	10
3.4	The comparison window	11
4.1	Separation of the query engine from the GUI	17
4.2	Output of the precision recall measurement device	17
5.1	One possible way from a corpus to models	21

Bibliography

- [1] *SWISH-RUN - Running Swish-e and Command Line Switches*. Manual for the Simple Web Indexing System for Humans, URL: swish-e.org/current/docs/SWISH-RUN.html#item_prog. cited on page(s) **19**
- [2] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990. cited on page(s) **21**
- [3] Daniel Fischer and Benedikt Grundmann. Alwis sourcecode, 2006. Published under the GNU General Public License (GPL). Available for download at <http://www.mpi-sb.mpg.de/~dfischer>. cited on page(s) **21, 24**
- [4] W. B. Frakes. Stemming algorithms. pages 131–160, 1992. cited on page(s) **20**
- [5] Benedikt Grundmann. Alwis – a visualisation tool for concept based retrieval schemes – theoretical foundations and models, 2006. This is the Bachelor thesis of my colleague. cited on page(s) **4, 5, 9, 11, 13, 14, 16, 19, 20, 21, 22, 23**
- [6] Thomas Hofmann. Probabilistic Latent Semantic Indexing. In *Proceedings of the 22nd Annual ACM Conference on Research and Development in Information Retrieval*, pages 50–57, Berkeley, California, August 1999. cited on page(s) **21**
- [7] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm, 2001. cited on page(s) **22**
- [8] Ian H. Witten, Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, May 1999. cited on page(s) **20**

Eidesstattliche Erklärung

Hiermit erkläre ich, Daniel Fischer, an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Daniel Fischer

Saarbrücken, den 13. März 2006