Class Search in NIO [*†]
(searching the web directory)
Documentation

Sebastian Bitzer
(sebastian.bitzer@uni-osnabrueck.de)

April 1, 2003

Introduction

Although the web directory in NIO provides an intuitive access to often used links a function to search not so frequently visited websites becomes with an growing web directory more and more important. Especially with an increasing number of groups within the web directory it is very difficult to keep track of them.

The class which is presented here (**search.class.php**) tries to satisfy this need for a search function which is structuring query results, too, and therefore provides a connection to the structure of the web directory.

This documentation will explain the way from the query to the structured output of the search results. This includes all subtasks such as finding keywords according to a query, valuating keywords by a distance measure, valuating links by values of related keywords and also evaluation of groups.

[*]this search originates in the seminar "Modelling user interests II" by Helmar Gust who also was deeply involved in designing this program

[†]NIO can be accessed via http://fuchur.informatik.uni-osnabrueck.de/nio

Contents

1 From query to valued keywords

In the web directory every web link and every group is connected to keywords which should describe the content of the link or group to some extent. Our search uses these keywords to find links and groups according to a query given from the user. In this section we describe how we get from a query to matching keywords.

The first step is to normalise the query, which is often done in information retrieval systems, so that also different word forms (e.g. verb and noun) and different writings of words (think of different writings for German umlauts) can be found. This is realised by a stemming. We also used to name this method 'lemmatisation', but if one looks at the definition of lemmatisation in more detail one will find that the method used here is more likely a stemming. The main difference between stemming and lemmatisation is that the result of stemming does not have to be a common word (but precisely a stem) whereas lemmatisation should result in a lemma corresponding to the input. Stemming therefore is easier to implement and more technical, because one just can cut common suffixes or affixes.

In addition a generalisation is needed to include, for example, combined words into search for keywords.

This generalisation in turn makes it necessary to evaluate each found keyword in comparison to the original search string.

1.1 Stemming

There are a number of sophisticated stemmers freely available in the internet [1], but it is very difficult to find one that will do exactly what you want it to do. It is an additional problem that within the web directory we have to handle German and English words at the same time. Thus we have temporarily implemented an own simple stemmer which works in the following way:

- handle German umlauts:
  substitute every occurrence of
  '&uuml;', '&ouml;', '&auml;', '&szlig;', 'ü', 'ö' ,'ä' ,'ß' (and capital letters) by

[1]e.g. the Porter stemmer which is available in several programming languages except for PHP from http://www.tartarus.org/∼martin/PorterStemmer/

'ue', 'oe', 'ae' or 'ss'
this is necessary because the web directory stores umlauts at present in html-form (e.g. '&auml;') but users will type in words with normal umlauts ('ä')

- minimal word length:
  to prevent words from being cut to meaningless short sequences of characters (e.g. 'singen' to 's') we introduced a minimal word length, i.e. every word has to keep a length of at least four characters after cutting of suffixes

- cutting suffixes:
  cut suffixes in exactly this order of appearance:
  'en', 'es', 'er', 'ung', 's', 'ing', 'ly', 'n', 'e'
  'en', 'es', 'er' and 'ung' are common German suffixes, shorter ones are before longer ones so that word constructions like 'Umgebungen' can be reduced appropriately
  's', 'ing', 'ly', 'n' and 'e' are mainly English suffixes

For the future one can think about a combination of existing stemmers for German and English or a real lemmatisation at least for English. Till then every query will be normalised by the above given algorithm which also dynamically writes a normalised version of each keyword into the database, if not already present there.

→ method *stem* source code in A.2

1.2 Generalisation

With a number of approximately 2000 web links the web directory is not too big. Therefore a further generalisation is needed in order to get an acceptably sized search result.

We decided to use a generalisation which reduces a word to its phonetic form. In particular this is soundex for which there exists an implementation in PHP.

Soundex produces a four-character string which codes the first letter of the word as the first character of the result. The other three characters build an integer corresponding to the phonetics of the input word (e.g. science will become 'S520'). Details are described by Donald Knuth in [1]. That means that soundex reduces every possible word to one of 26000 such codings whereby, for example, words starting with 'x' are quite rare.

This results in a strong robustness against spelling mistakes, but semantically completely unrelated words like 'Maus' and 'Mausoleum' will nevertheless not be matched, because they get different phonetic codings. On the other hand combined words like 'cognitive' and 'cognitive architecture' will be matched with this method. For an explanation why this is sometimes necessary see 1.4 .

→ method *parse_query* source code in A.1

1.3 Evaluation of keywords by distance measure

A request to the database with a stemmed and 'soundexed' query will yield a lot of keywords that do not correspond to the spelling of the original query. The question then is, of course, how good these keywords match to the query.

We introduced a distance measure between words to handle this problem. The core of this distance measure is the Levenshtein distance between two strings. It calculates the minimum number of characters you have to delete, insert or substitute to transform string1 into string2. The Levenshtein distance was first published by Vladimir Levenshtein [2]. We used the recent PHP implementation of this distance measure with equal costs for deletion, insertion and substitution. In addition we normalised the measure to values between 0 and 1 where 1 means that the two strings match completely [2]. This is done by subtracting the division of the result of the Levenshtein distance by the length of the longest string [3] from 1:

$$distance(string1, string2) = 1 - \frac{levenshtein(string1, string2)}{max(length(string1), length(string2))}$$

We just compare stems with the Levenshtein distance, but before we do so we check, if the found keyword is a substring of the original query and if it is, it gets the highest value 1 [4]. If the stem of the keyword matches to the stem

[2]in view of an evaluation with better values = higher values

[3]which is equal to the supremum of this Levenshtein distance

[4]again the values are normalised to be between 0 and 1

of a queried word, it gets the value 0.9, else the Levenshtein distance between the stems of keyword and queried word is calculated and gets a value of at most 0.8. This is easily calculated by:

$$value(keyword) = 0.8 \cdot distance(stem(keyword), stem(queried\ word)).$$

→ method *get_key_value* source code in A.4

1.4 Putting all together

So what happens exactly, when a query is passed to the system?

At first the query is parsed into single words and is standardised to lower letters. Then every such word is stemmed and soundex-codes for these stems are calculated. Every keyword whose soundex-code matches one of the codes of the query is fetched from the database. In the web directory a keyword can also be a group of words, but the query is always parsed into single words. Therefore it is useful to have a generalisation, like soundex, on hand that also allows to find a group of words containing a queried word (see 1.2). Each keyword is then evaluated in comparison to each queried word as explained above. The maximum of these comparisons is taken as the value of the keyword.

Figure 2 gives a graphical overview over the algorithm stated so far on the basis of the sample query 'Cognitive Science'.

→ method *set_vars* source code in A.7 on page 21/top of 22

2 From valued keywords to a sorted link list

With keywords as the semantical description of a link it is possible to access web links that should have a relation to the query. Again it has to be evaluated how strong this relation is. In this section we describe by which mechanism this is done.

The starting point for this is that we already have valued keywords and have used them to get all links that are associated with them.

2.1 Principle mechanism

The main principle behind the mechanism we utilise here is the idea that a link has a stronger relation to the query, if there are more keywords [5] that are associated with that link. If you have that, all that has to be added then is just the consideration of different values for keywords.

So what we do in principle is the following:

1. go through all links
2. store link
3. for every keyword associated with link add a value calculated from the value of the keyword to the value of the stored link
4. sort links according to their values

2.2 Mechanism in detail

Of course, it is not exactly as simple as one could think now especially the value calculated from the value of a keyword needs to be discussed further, but we will see.

We get from the database after having requested for links connected to the keywords a list of link-keyword combinations in which one row contains a link with one of its connected keywords. That means that we get exactly as much rows in this list as there are keyword associations to links:

$$rows = \sum_{i=1}^{nr.links} nr.keys_i$$

whereby a link in this context is an internally generated and unique id-number corresponding to a web link (link-id) and the keywords are still just the keywords which were found as described above (1).

Then we go through this list and write the links into an associative array with links (ids) as keys and values calculated from corresponding values of keywords as values of the array. Every additional occurrence of a link in the

[5] which were found before with the algorithm presented in section 1

list then results in an increase of its value in the array dependent on the value of the corresponding keyword in this row of the list.

The first value of a link is initialised with the half of the value of the corresponding keyword:

$$v_0 = \frac{v_{key}}{p_0}, \qquad p_0 = 2.$$

With every subsequent occurrence of the same link a normalised term is added to the current value of the link:

$$v = v_{cur} + \frac{1 - v_{cur}}{p_1} \cdot v_{key}^{p_2}, \qquad p_1 = 2,\ p_2 = 2.$$

This normalisation again restricts values of links to lie in between 0 and 1 [6]. Thereby the dependency of the influence of the key value on the "reversed" current value leads to an asymptotic behaviour of the link value towards 1. That means that with increasing value the influence of further keywords reduces. The parameters p_1 and p_2 are set to 2 at present. Changing p_1 will result in a different convergence velocity [7] whereas p_2 adjusts the influence of key values. With an increasing p_2 good values of keys become more and more better and bad values become more and more worse.

When this is done, a value should result that gives a more or less good measure for the strength of the relation between query and link. Thus the right adjustment of all parameters is indispensable for a good search result and the system should be continuously observed to ensure that.

→ method *set_vars* source code in A.7 on page 22

2.3 Alternative scheme

We first pursued a slightly different scheme which we gave up because of reasons explained in section 4.

This scheme included a sorting of the valued keywords. After having done so we took the first keyword to fetch links associated with it. These links were evaluated immediately. Then the second best keyword delivered the next links (or maybe the same) and so on.

[6]like it is done for values of keys

[7]increasing will slow it down, decreasing will accelerate it

The advantage of this approach is that one can cut off keywords that seem to be too weak in their value to meaningfully contribute to the values of links. For example we said that all keywords with a value worse than $\frac{1}{7}$ of the maximum key value could be cut off. This is another parameter that has to be adjusted.

Figure 3 shows this scheme (that is nevertheless quite similar to the one presented above, if you do not cut off keywords, you even have exactly the same results as above) on basis of the 'Cognitive Science' example.

3 Structuring by including groups in search

The web directory provides a built in structure for categorisation of web links. It would be a shame, if one does not use this to organise the search results. We explain in this section how we use groups to do that.

Again the relations between groups and query have to be evaluated, but also the relations between groups and links are important.

3.1 Evaluating groups

For evaluation of groups we fall back on the evaluation of links in double respects: We use the same method as for the evaluation of links but with link values in place of key values (see 2.2)

So the groups build a third abstraction layer of the query with a lot of processing happened in the preceding time. It is therefore no wonder that search results for groups mostly look very good in relation to the query.

On the other hand links have a very strong impact on the search for groups when using this method. This is a problem when one searches, for example, for 'php open source'. Links connected to 'php' strongly outnumber links connected to 'open source'. So group 'open source' gets a rather small value (in comparison to group 'php programmieren'). To handle this we wrote a method that searches for groups directly without considering links. This happens in exactly the same way as it is done when searching for links (see sections 1 and 2). However, this method is not in use yet. The Problem of this method is that the more general keywords connected to groups can not be found when searching for more special things and therefore these groups can be found neither although they maybe contain these special things.

→ method *search_groups* source code in A.6

In addition we wrote another method for searching groups for the process of entering new links in the web directory. This method takes a list of valued keywords and returns a sorted list of groups which are connected to exactly these keywords. In between lies again the same evaluation algorithm as described in 2.2. The difference to the method mentioned above is that keywords are not generalised in this method.

→ method *search_groups_int* source code in A.5

3.2 Allocation of links to groups

We decided to show the group-independent, overall search result first while at the same time presenting found groups so that the user can change to the special search result of a group, if he thinks that he will find his wanted web link easier there.

Within the process of this search no new allocations of links to groups are done. We rely on the existing connections between links and groups in the web directory. Not until a user has clicked on a group in the search result these connections are considered for displaying links. Then just these links are shown which are associated with the group clicked. Thereby the values of the links are not changed. Since links in the web directory can be connected to several groups it is possible that links appear in different groups of the search result, too.

Figure 1 illustrates the allocation of links to groups when a group was clicked in the search result.

→ method *set_vars* source code in A.7 on page 23

4 Performance matters

We had to give up the alternative evaluation scheme mentioned in 2.3, because we experienced severe performance problems when sending a lot of small requests to the database.

To solve this problem we combined all requests to one, i.e. that we now formulate just one complicated request for the database which then returns a list containing all information we need. One entry in this list consists of a link-id a keyword (with id-number, word and lemma/stem) and a group (also just an id-number).

link	*keyword*			*group*
id	*id*	*word*	*lemma*	*id*
1196	2884	*'science'*	*'scienc'*	115
1196	5053	*'cognition'*	*'cognitio'*	115
1196	5053	*'cognition'*	*'cognitio'*	69
...				

This led to an improvement of processing speed on the part of the database. On the other hand, one easily sees that the size of this list can get quite large since one link may have many entries in it (as much as combinations of different keywords and groups). The size of this list could be calculated in the following way:

$$rows = \sum_{i=1}^{nr.links} nr.keys_i \cdot nr.groups_i.$$

Unfortunately this combination of requests did not solve our performance problems entirely. For optimisation we worked out so far that we just need to run through the whole list twice: once for evaluation of keys and links and thereafter once for evaluation of groups [8]. Nevertheless the system still needs six seconds on average to respond to a query what is not acceptable for a website. We hope that the change to a faster machine on which NIO is running further improves this response time.

→ method *set_vars* source code in A.7

5 Conclusion

We introduced here a sophisticated method for searching the web directory within NIO.

Although this method works already quite good there are still some options left open for improving it. Especially the right adjustment of all parameters could bring an improvement of the search result. One also can think

[8]which needs the links to be evaluated already

of additional features like consulting link titles and comparing them to the query to make search better.

This documentation should have made one thing clear: our search method is heavily dependent on the allocation of keywords to links and groups. The more meaningful associations between keywords and links exist the better is the quality of our search result. Therefore a well maintained and manifold system of keywords in NIO helps to assure useful search results, too.

References

[1] Knuth, D.: *The Art Of Computer Programming, vol. 3: Sorting And Searching*, Addison-Wesley, pp. 391-392, 1973.

[2] Levenshtein, V. I.: *Binary codes capable of correcting deletions, insertions and reversals*, Doklady Akademii Nauk SSSR 163(4) p845-848, 1965
also Soviet Physics Doklady 10(8) p707-710, Feb 1966.

A Source code

A.1 parse-query

```
/**
 * takes a query cuts it into single words and stems these,
 * stems are generalised by soundex function; sets query instance variables
 *
 * @param STRING a query to parse
 * @author  Sebastian Bitzer <sebazi@gmx.net>
 * @version 28.02.2003 [zib]
 */
function parse_query($query)
{
   $this->query = strtolower($query);
   $this->a_query = preg_split("/[\s,]+/",$this->query);
   if (sizeof($this->a_query)>1 && strlen($this->query)>10)
      $this->a_query[] = $this->query;

   for ($i=0; $i<sizeof($this->a_query);$i++)
   {
      $lemma = $this->lemma($this->a_query[$i]);
      $this->a_query_lemma[] = $lemma;
      if ($i==0)
         $s_query = "'".soundex($lemma)."'";
      else
         $s_query.= ",'".soundex($lemma)."'";
   }
   $this->query_sound = $s_query;
}
```

A.2 stem

```
/**
 * produces a stem from a given word through cutting some word endings
 * and normalising german umlauts (to ae, ue, oe, ss)
 *
 * @param STRING a word
 * @return STRING stemmed word
 * @author Sebastian Bitzer <sebazi@gmx.net>
 * @version 27.02.2003 [zib]
 */
function stem($word)
{
   //suffixes that are deleted during stemming
   $a_suffix = array('en','es','er','ung','s','ing','ly','n','e');
   //substrings that will be replaced during normalisation
   $a_string_subst1 = array('&auml;','&uuml;','&ouml;',
                            '&Auml;','&Uuml;','&Ouml;','&szlig;',
                            '' ,'' ,'' ,'' ,'' ,'' ,'');
   //replacements
   $a_string_subst2 = array('ae'    ,'ue'    ,'oe'    ,
                            'ae'    ,'ue'    ,'oe'    ,'ss'     ,
                            'ue','oe','ae','ue','oe','ae','ss');

                                          //easiest stem: word
   $stem = $word;
                                          //replace umlauts
   foreach($a_string_subst1 as $i => $string)
      $stem = str_replace($string,$a_string_subst2[$i],$stem);
                                          //just if word has more than 4 letters
   if (strlen($stem)>4)
   {                                      //go through suffixes
      foreach($a_suffix as $suffix)
      {                                   //if result has more than 4 letters
         if (strlen($stem)-strlen($suffix)>4 &&
             $suffix == substr($stem,-strlen($suffix)))
                                          //cut special ending
            $stem = substr($stem,0,-strlen($suffix));
      }
   }
   return $stem;
}
```

A.3 distance

```
/**
 * calculates distance between two words, against normal definition of distances
 * 0 is largest distance (by means of standardization throughout class)
 *
 * @param STRING first word
 * @param STRING second word
 * @return REAL estimated distance of words, 1 is best, 0 is worst
 * @author Sebastian Bitzer <sebazi@gmx.net>
 * @version 27.02.2003 [zib]
 */
function distance($word1, $word2)
{
   return 1 - levenshtein($word1, $word2)
              / max(strlen($word1),strlen($word2));
}
```

A.4 get-key-value

```
/**
 * evaluates a key in comparison to query, uses a distance measure
 * between stems
 *
 * @param ARRAY containing key as original string and lemma
 *        array(['wort']=>STRING
 *              ['lemma']=>STRING)
 * @return REAL estimated value of key, 1 is best, 0 is worst
 * @author Sebastian Bitzer <sebazi@gmx.net>
 * @version 27.02.2003 [zib]
 */
function get_key_value($line)
{
   $o_db_keys = new nio_simple_data(NIODB_DBNAME,NIODB_CONNECT);
   $o_db_keys->NIO_Connect('w','s');

   foreach($this->a_query_lemma as $lemma)
   {
      if (strstr($this->query,$line['wort']) != FALSE)
      {
         $o_db_keys->Update(array('lemma'=>$this->lemma($line['wort'])),
                            'schluesselwortnr = '.$line['schluesselwortnr']);
         $key_value = 1;
         break;
      }
      else
      {
         $line['lemma'] = $this->lemma($line['wort']);
         $o_db_keys->Update(array('lemma'=>$line['lemma']),
                            'schluesselwortnr = '.$line['schluesselwortnr']);
         if ($lemma == $line['lemma'])
            $key_value = max($key_value,0.9);
         else
         {
            $key_value = max($key_value,
                            0.8*$this->distance($lemma,$line['lemma']));
         }
      }
   }
   return $key_value;
}
```

A.5 search-groups-int

```
/**
 * similar to search_groups, but gets an array of evaluated keywords:
 * evaluates and sorts groups according to the values of entered keywords
 * and quantity of connections between groups and keys
 *
 * @param ARRAY valued keywords: array([keyword1]=>value,
 *                                    [keyword2]=>value,
 *                                    ...)
 * @return ARRAY groups sorted according to their evaluation,
 *               array([group_id1]=>highest value,
                       [group_id2]=>second highest value,
                       ...)
 * @author  Sebastian Bitzer <sebazi@gmx.net>
 * @version 31.03.2003 [zib]
 */
function search_groups_int($a_query)
{
   if ($a_query)
   {
      $this->a_query = $a_query;
      foreach($a_query as $word=>$wvalue)
      {
         $s_query.="'".$word."',";
      }
      $s_query = substr($s_query, 0, (strlen($s_query)-1));
   }
   else
      return 'can\'t search, no query';

   $o_db_data = new nio_data(NIODB_DBNAME,NIODB_CONNECT);
   $o_db_data->NIO_Connect();

   $a_query_data = $o_db_data->Execute("
                 select      gruppe.idnr as gruppennr,
                             gruppe.bezeichnung as titel,
                             schluessel.idnr as schluesselwortnr,
                             schluessel.bezeichnung as wort,
                             schluessel.lemma
                 from        nio_schluessel schluessel,
                             nio_gruppe gruppe,
                             nio_gruppe_schluessel gs
                 where       schluessel.bezeichnung in ($s_query)
                        and  schluessel.domaene='w'
                        and  schluessel.idnr=gs.schluessel_idnr
                        and  gs.gruppe_idnr not in (1,2,3)
                        and  gs.gruppe_idnr=gruppe.idnr
                        and  gruppe.domaene='w'
                 order by    gruppennr, schluesselwortnr
                  ");
   $o_db_data->NIO_Disconnect();

   if(is_array($a_query_data))
   {
      foreach ($a_query_data as $line)
         {
         $group_id = $line['gruppennr'];
         $key_id = $line['schluesselwortnr'];
         $value = $a_query[$line['wort']];
         if ($a_value_groups[$group_id])
         {
            $current_value = $a_value_groups[$group_id]['value'];
            $a_value_groups[$group_id]['value'] = $current_value
                                              +  (1-$current_value)
                                                 /   2
                                                 *   pow($value,2);
         }
         else
            $a_value_groups[$group_id] = array(
                           'value'=>$value/2,
                           'titel'=>$line['titel']
                           );
      }

      asort($a_value_groups);
      $a_sorted_groups = array_reverse($a_value_groups,TRUE);

      return $a_sorted_groups;
   }
   else
      return 'no groups found';
}
```

A.6 search-groups

```
/**
 * similar to search_all, but works completely without consideration of links:
 * evaluates keys and according to this evaluation and quantity of connections
 * between groups and keys evaluates and sorts groups
 *
 * @param STRING a query
 * @return ARRAY groups sorted according to their evaluation,
 *               array([group_id1]=>highest value
                       [group_id2]=>second highest value
                       ...)
 * @author  Sebastian Bitzer <sebazi@gmx.net>
 * @version 28.02.2003 [zib]
 */
function search_groups($query)
{
   if (!$this->query)
      if ($query)
         $this->parse_query($query);
      else
         return 'can\'t search, no query';

   $o_db_data = new nio_data(NIODB_DBNAME,NIODB_CONNECT);
   $o_db_data->NIO_Connect();

   $a_query_data = $o_db_data->Execute("
                 select      gruppe.idnr as gruppennr,
                             schluessel.idnr as schluesselwortnr,
                             schluessel.bezeichnung as wort,
                             schluessel.lemma
                 from        nio_schluessel schluessel,
                             nio_gruppe gruppe,
                             nio_gruppe_schluessel gs
                 where       schluessel.phonetik in ($this->query_sound)
                        and  schluessel.domaene='w'
                        and  schluessel.idnr=gs.schluessel_idnr
                        and  gs.gruppe_idnr not in (1,2,3)
                        and  gs.gruppe_idnr=gruppe.idnr
                        and  gruppe.domaene='w'
                 order by    gruppennr, schluesselwortnr
                       ");
   $o_db_data->NIO_Disconnect();

   foreach ($a_query_data as $line)
   {
      $group_id = $line['gruppennr'];
      $key_id = $line['schluesselwortnr'];
      if (!$a_value_keys[$key_id])
      {
         $a_value_keys[$key_id] = $this->get_key_value($line);
      }
      $value = $a_value_keys[$key_id];
      if ($a_value_groups[$group_id])
      {
         $a_value_groups[$group_id] = $a_value_groups[$group_id]
                                   +  (1-$a_value_groups[$group_id])
                                      /  2
                                      *  pow($value,2);
      }
      else
         $a_value_groups[$group_id] = $value/2;
   }

   asort($a_value_groups);
   $a_sorted_groups = array_reverse($a_value_groups,TRUE);

   return $a_sorted_groups;
}
```

A.7 set-vars

```
/**
 * parses query, fetches result delivered by databank for soundex-normalized
 * query, evaluates keys, with that evaluates links and sorts them, then
 * evaluates groups and sorts them, initializes new http_vars used for pagelinks
 *
 * @param OBJECT a databank connection
 * @param STRING $postfix Postfix var for httpvars
 *                        (desktop-style: different group-pagelinks)
 * @return ARRAY array([0]=>array([link_id1]=>highest value
                                  [link_id2]=>second highest value
                                  ...)
                       [1]=>array([group_id1]=>array(['value']=>highest value)
                                  ...
                                  [group_id of current group]=>
                                       array(['value']=>value
                                             ['links']=>links in group)
                                  ...))
 * @author Sebastian Bitzer <sebazi@gmx.net>
 * @version 27.02.2003 [zib]
 */
function set_vars($o_db_data,$postfix=NULL)
{
   $this->parse_query($this->http_vars['query']);
   if ($this->query == '')
   {
      $this->num_links = 0;
      return array(NULL,NULL);
   }

   $a_query_data = $o_db_data->Execute("
                  select      link.idnr as linknr,
                              gruppe.idnr as gruppennr,
                              schluessel.idnr as schluesselwortnr,
                              schluessel.bezeichnung as wort,
                              schluessel.lemma
                  from        nio_verweis link,
                              nio_schluessel schluessel,
                              nio_gruppe gruppe,
                              nio_schluessel_verweis sv,
                              nio_gruppe_verweis gv
                  where       schluessel.phonetik in ($this->query_sound)
                         and  schluessel.domaene='w'
                         and  schluessel.idnr=sv.schluessel_idnr
                         and  sv.verweis_idnr=link.idnr
                         and  link.domaene='w'
                         and  link.idnr=gv.verweis_idnr
```

```
                      and  gv.gruppe_idnr not in (1,2,3)
                      and  gv.gruppe_idnr=gruppe.idnr
                      and  gruppe.domaene='w'
               order by    linknr, schluesselwortnr, gruppennr
                     ");

foreach ($a_query_data as $line)
{
   $link_id = $line['linknr'];
   $key_id = $line['schluesselwortnr'];
   if (!$a_value_keys[$key_id])
   {
      $a_value_keys[$key_id] = $this->get_key_value($line);
   }
   if ($link_id != $prev_link_id or $key_id != $prev_key_id)
   {
      $value = $a_value_keys[$key_id];
      if ($a_value_links[$link_id])
      {
         $a_value_links[$link_id] =
                     $a_value_links[$link_id]
                  +  (1-$a_value_links[$link_id])
                     /  2
                     *  pow($value,2);
      }
      else
         $a_value_links[$link_id] = $value/2;
   }
   $prev_link_id = $link_id;
   $prev_key_id = $key_id;
}

if ($this->http_vars['grp'])
{
   $group = $this->http_vars['grp'];
}
else
{
   asort($a_value_links);
   $a_sorted_links = array_reverse($a_value_links,TRUE);
}

/*---------------------------GROUPS------------------------------*/
foreach($a_query_data as $line)
{
   $group_id = $line['gruppennr'];

   if ($group)
   {
      $value = $a_value_links[$line['linknr']];
      if ($group == $group_id)
      {
         $a_value_groups[$group_id]['links'][$line['linknr']] = $value;
      }
   }
   else
      $value = $a_sorted_links[$line['linknr']];

   if ($a_value_groups[$group_id])
   {
      $a_value_groups[$group_id]['value'] =
                     $a_value_groups[$group_id]['value']
                  +  (1-$a_value_groups[$group_id]['value'])
                     /  2
                     *  pow($value,2);
   }
   else
      $a_value_groups[$group_id]['value'] = $value / 2;
}

function compare_value_groups($a_key_value1, $a_key_value2)
{
   if ($a_key_value1['value'] == $a_key_value2['value'])
      return 0;
   return $a_key_value1['value'] > $a_key_value2['value'] ? -1 : 1;
}
```

```
   if ($group)
   {
      $a_tmp = $a_value_groups[$group]['links'];
      asort($a_tmp);
      $a_sorted_links = array_reverse($a_tmp,TRUE);
   }

   /*-----------------------------HTTP_VARS-------------------------------*/

   if ($this->http_vars['position'.$postfix])
   {
      $this->position=$this->http_vars['position'.$postfix];
   }
   else
   {
      $this->position = 0;
   }
                                    // store size of search result
   $this->num_links = sizeof($a_sorted_links);
                                    // if "entries per page" is submitted
   if ($this->http_vars['epp'.$postfix])
                                    // store it in instance variable
      $this->epp=$this->http_vars['epp'.$postfix];
   else                             // else
      $this->epp = 10;              // set epp to default (10)

   return array ($a_sorted_links,$a_value_groups);
}
```

A.8 search-all

```
/**
 * if $this->query is not set it is set to query-parameter, calls
 * set_vars, delivers result according to parameters
 *
 * @param BOOLEAN specifies if groups should be delivered
 * @param BOOLEAN specifies if links should be delivered
 * @param STRING a query; has just effect, if $this->query is not set!
 * @return ARRAY from set_vars, if $groups and $links: array([0]->links
 *                                                              [1]->groups)
 *                              if $groups: array(groups)
 *                              if $links: array(links)
 * @author  Sebastian Bitzer <sebazi@gmx.net>
 * @version 28.02.2003 [zib]
 */
function search_all($goups=TRUE,$links=TRUE,$query=null)
{
   if (!$this->query)
      if ($query)
         $this->query = strtolower($query);
      else
         return 'can\'t search, no query';

   $o_db_data = new nio_data(NIODB_DBNAME,NIODB_CONNECT);
   $o_db_data->NIO_Connect();
   $a_result=$this->set_vars($o_db_data);
   $o_db_data->NIO_DISCONNECT;

   if ($groups and $links)
      return $a_result;
   else if ($groups)
      return $a_result[1];
   else if ($links)
      return $a_result[0];
   else
      return 'no return specified, when calling search_all!!!';
}
```

B Graphical overview

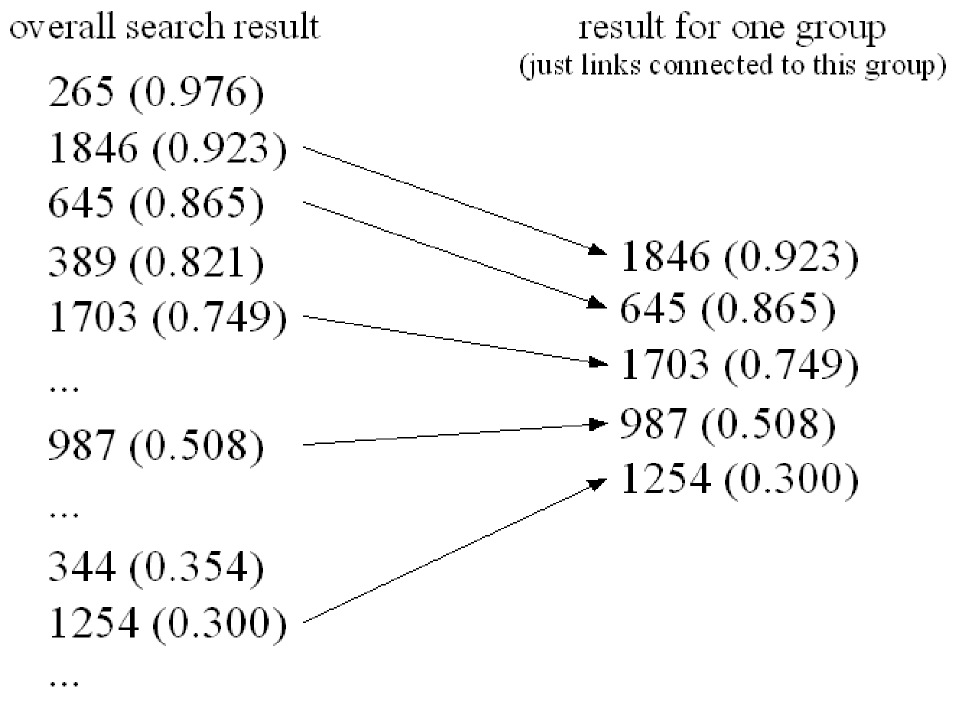

Figure 1: Allocation of links to a group: link-id(value)

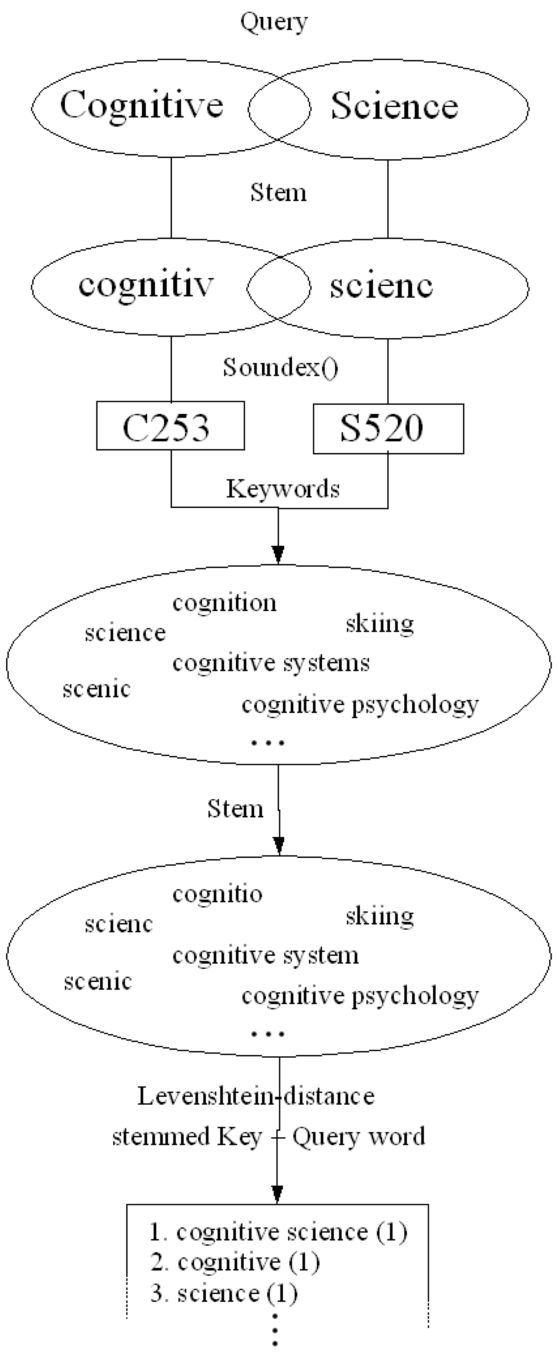

Figure 2: From query to valued keywords

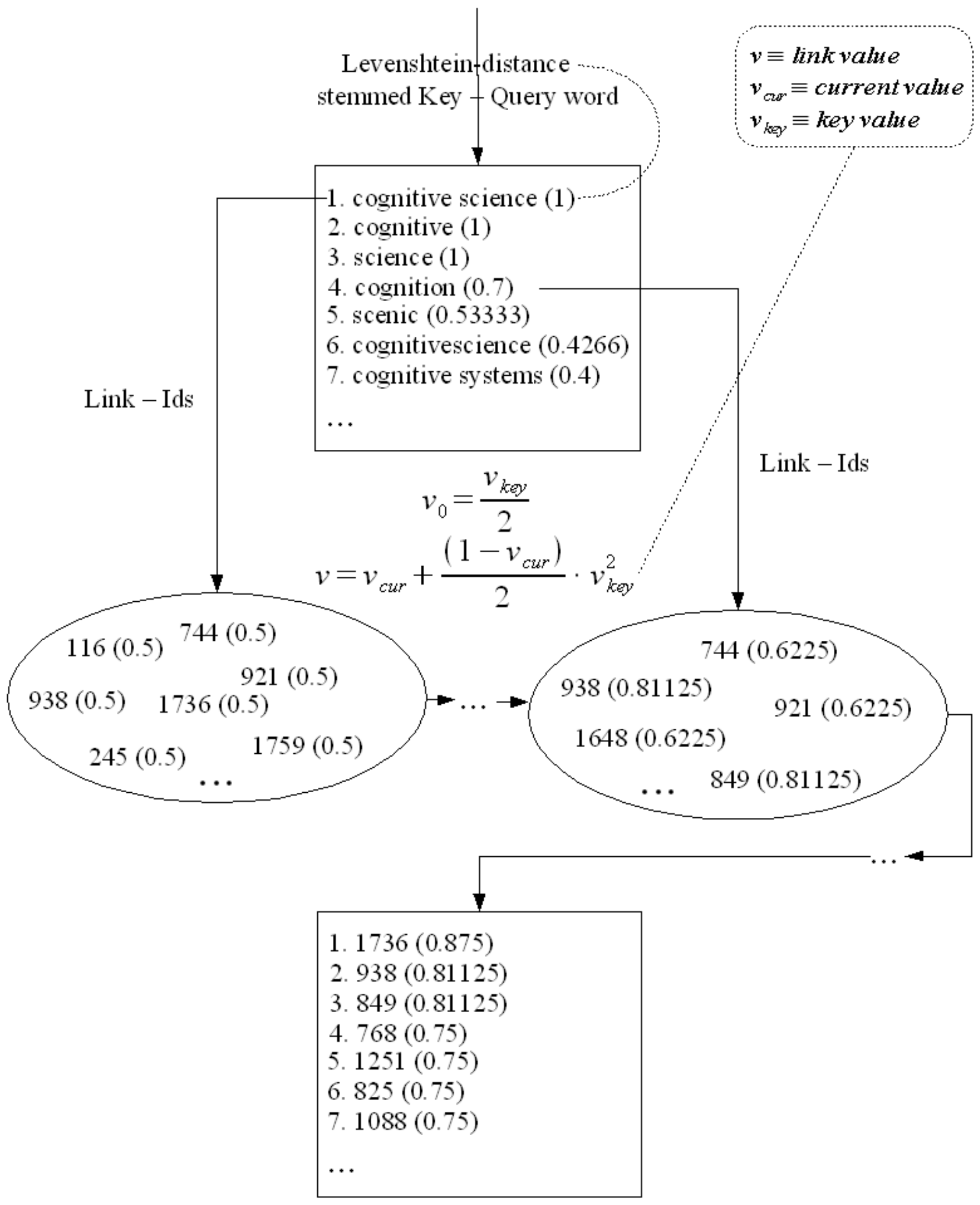

Figure 3: From valued keywords to a sorted link list (alternative scheme)